Redes Neuronales

Matias Vera

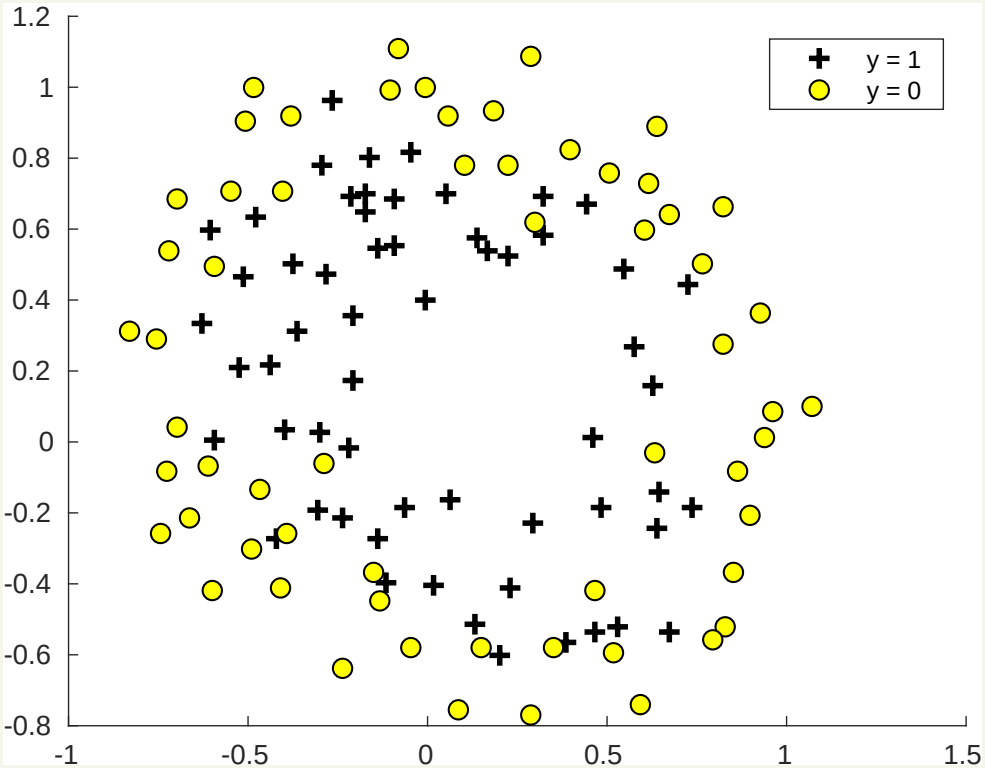
Taller de Procesamiento de Señales

Agenda

1. Redes Neuronales y el Deep World
2. Entrenamiento de Redes Neuronales Profundas
3. Detalles de Deep Learning
4. Regularización
5. Redes Convolucionales
6. Redes Recurrentes

Las soluciones lineales pueden ser insuficientes

¿Como reducir el riesgo empírico?



Aproximación Universal

Teorema

Sea $\mathcal{A} \subset \mathbb{R}^m$ un conjunto cerrado y acotado y sea $g : \mathbb{R} \to \mathbb{R}$. Para cada función continua $f : \mathcal{A} \to \mathbb{R}$ y $\epsilon > 0$ existe un $n \in \mathbb{N}$ y $\{\mathbf{w}_i, b_i, \alpha_i\}_{i=1}^n$ con $\mathbf{w}_i \in \mathbb{R}^m$ y $b_i, \alpha_i \in \mathbb{R}$ para todo $i = 1, \cdots, n$ tales que

$$\left| f(\mathbf{x}) - \sum_{i=1}^{n} \alpha_i g(\mathbf{w}_i^T \mathbf{x} + b_i) \right| < \epsilon \qquad \forall\, \mathbf{x} \in \mathcal{A}$$

si y solo si g no es un polinomio.

Necesito funciones no lineales!

Leshno and Schocken 1993: "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function".

Aproximación Universal

Teorema

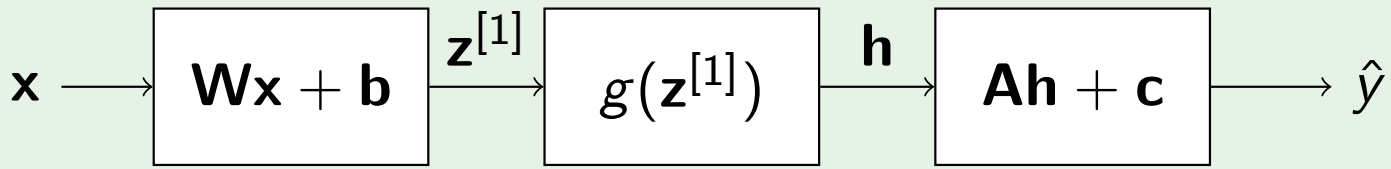
Sea $\mathcal{A} \subset \mathbb{R}^m$ un conjunto cerrado y acotado y sea $g : \mathbb{R} \to \mathbb{R}$. Para cada función continua $f : \mathcal{A} \to \mathbb{R}$ y $\epsilon > 0$ existe un $n \in \mathbb{N}$ y $\{\mathbf{w}_i, b_i, \alpha_i\}_{i=1}^n$ con $\mathbf{w}_i \in \mathbb{R}^m$ y $b_i, \alpha_i \in \mathbb{R}$ para todo $i = 1, \cdots, n$ tales que

$$\left| f(\mathbf{x}) - \sum_{i=1}^{n} \alpha_i g(\mathbf{w}_i^T \mathbf{x} + b_i) \right| < \epsilon \qquad \forall\, \mathbf{x} \in \mathcal{A}$$

si y solo si g no es un polinomio.

Necesito funciones no lineales!

Regresión



Leshno and Schocken 1993: "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function".

Aproximación Universal

Teorema

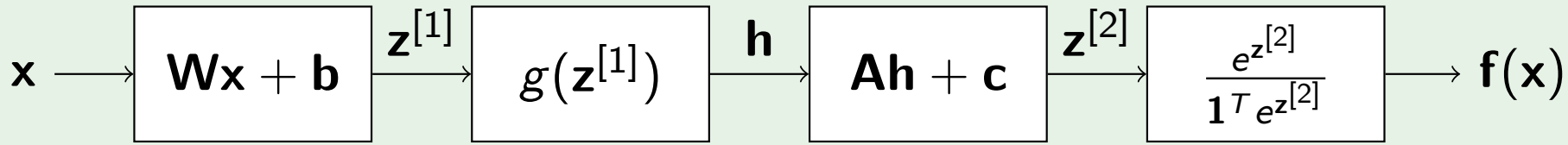
Sea $\mathcal{A} \subset \mathbb{R}^m$ un conjunto cerrado y acotado y sea $g : \mathbb{R} \to \mathbb{R}$. Para cada función continua $f : \mathcal{A} \to \mathbb{R}$ y $\epsilon > 0$ existe un $n \in \mathbb{N}$ y $\{\mathbf{w}_i, b_i, \alpha_i\}_{i=1}^n$ con $\mathbf{w}_i \in \mathbb{R}^m$ y $b_i, \alpha_i \in \mathbb{R}$ para todo $i = 1, \cdots, n$ tales que

$$\left| f(\mathbf{x}) - \sum_{i=1}^{n} \alpha_i g(\mathbf{w}_i^T \mathbf{x} + b_i) \right| < \epsilon \qquad \forall\, \mathbf{x} \in \mathcal{A}$$

si y solo si g no es un polinomio.

Necesito funciones no lineales!

Clasificación



Leshno and Schocken 1993: "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function".

Aproximación Universal

Teorema

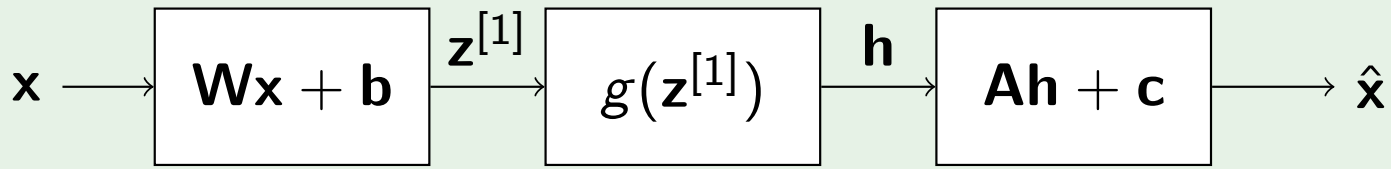
Sea $\mathcal{A} \subset \mathbb{R}^m$ un conjunto cerrado y acotado y sea $g : \mathbb{R} \to \mathbb{R}$. Para cada función continua $f : \mathcal{A} \to \mathbb{R}$ y $\epsilon > 0$ existe un $n \in \mathbb{N}$ y $\{\mathbf{w}_i, b_i, \alpha_i\}_{i=1}^n$ con $\mathbf{w}_i \in \mathbb{R}^m$ y $b_i, \alpha_i \in \mathbb{R}$ para todo $i = 1, \cdots, n$ tales que

$$\left| f(\mathbf{x}) - \sum_{i=1}^{n} \alpha_i g(\mathbf{w}_i^T \mathbf{x} + b_i) \right| < \epsilon \qquad \forall\, \mathbf{x} \in \mathcal{A}$$

si y solo si g no es un polinomio.

Necesito funciones no lineales!

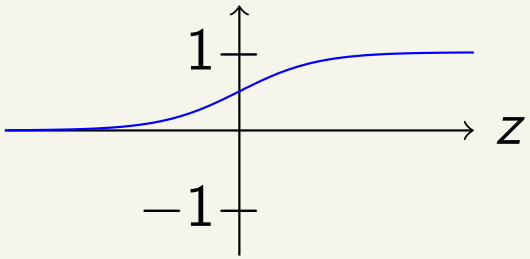
Autoencoder



Leshno and Schocken 1993: "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function".

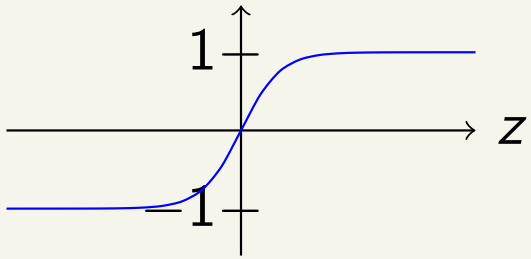
Funciones de Activación g

$$g(z) = \frac{1}{1+e^{-z}}$$



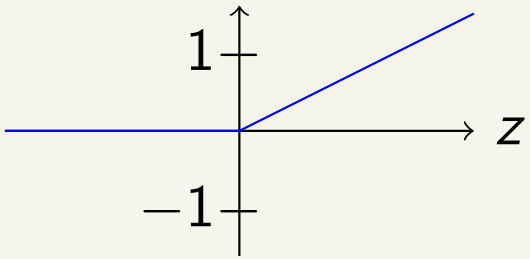
Sigmoide

$$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$



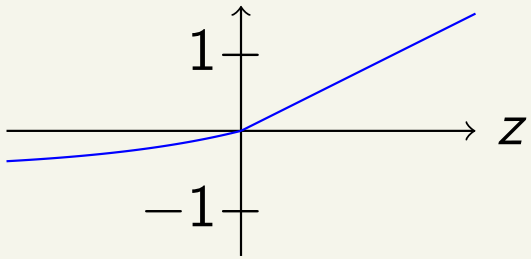
Tangente hiperbólica

$$g(z) = \max\{z, 0\}$$



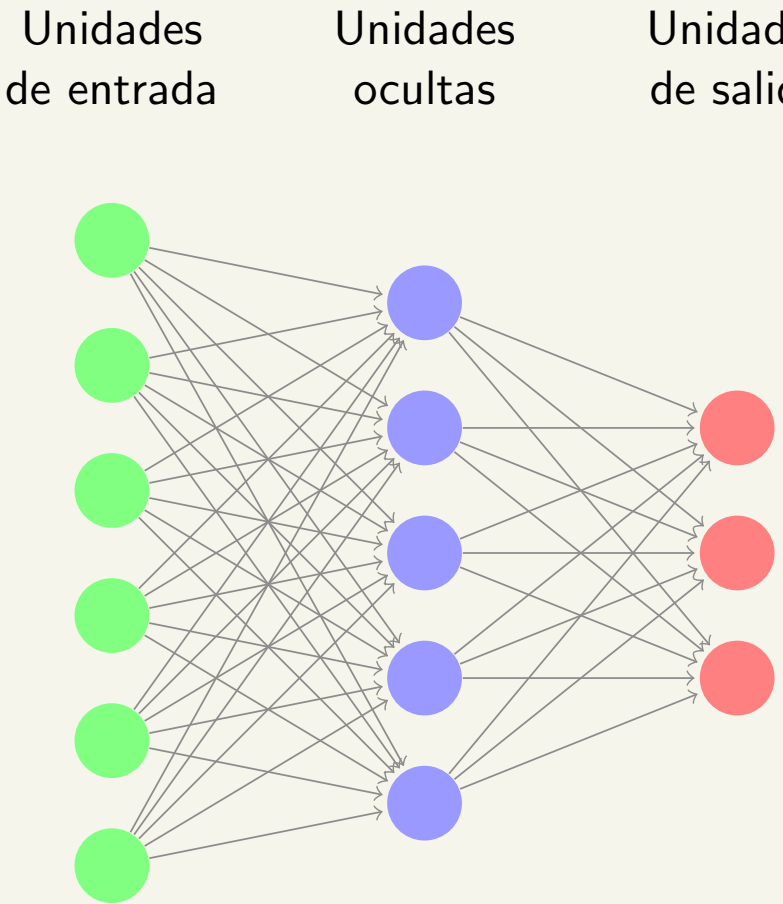
ReLU

$$g(z) = \begin{cases} z & z \geq 0 \\ \alpha(e^{z}-1) & z < 0 \end{cases}$$



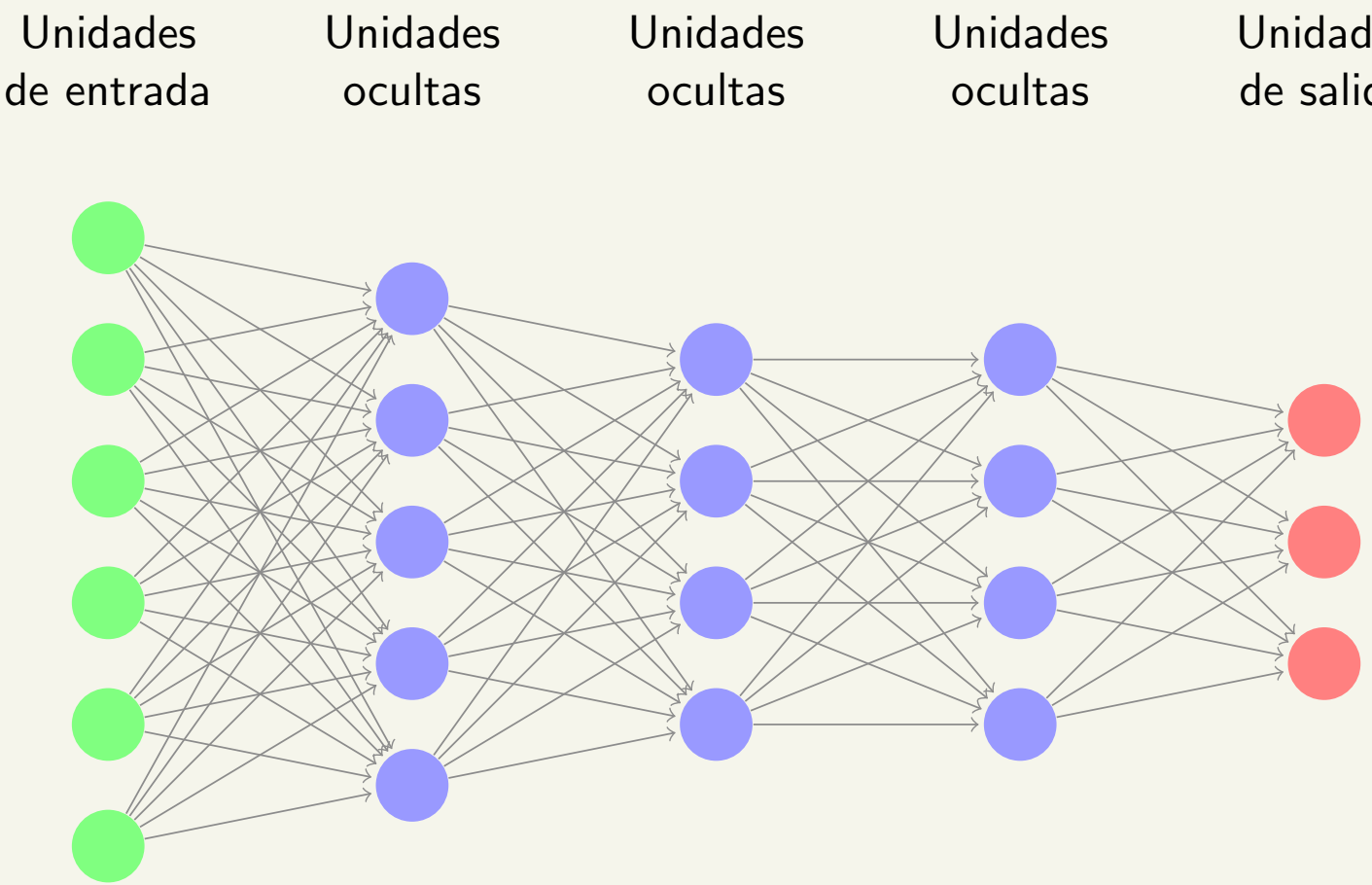
$\alpha-$ELU

Deep Learning



Red neuronal básica

Deep Learning



Deep Learning Architecture

¿Por que muchos layers?

Los teoremas de aproximación universal nos hablan de la existencia de parámetros, no como elegirlos ni cuantas unidades ocultas usar. Algunas ventajas del deep-world:

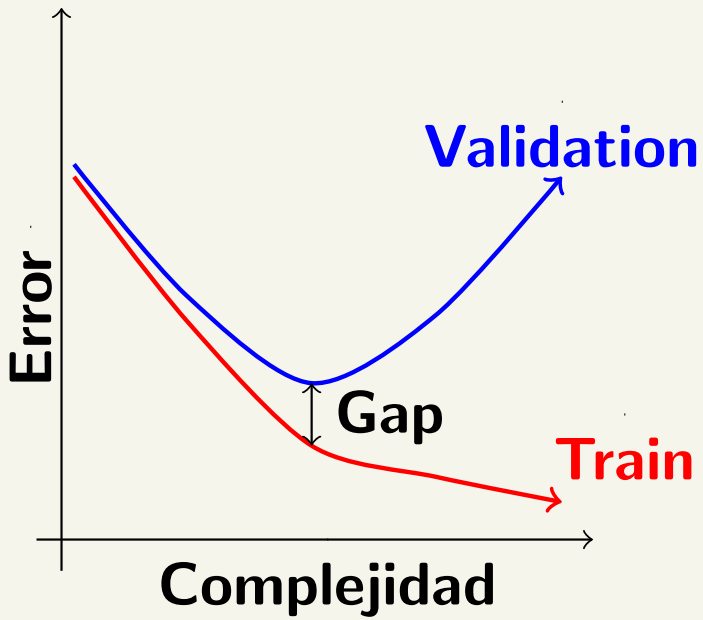
- Las redes profundas suelen ser más eficientes en la etapa del entrenamiento y tienen mejores propiedades de generalización.
- La cantidad de unidades ocultas necesarias para aproximar suele decaer exponencialmente con la profundidad de la red.
- En algunos casos se puede demostrar que la cantidad de unidades ocultas necesarias crece exponencialmente con la dimensión de la entrada.

Características del Deep-world

- Big Data (A partir de 30K o 40K muestras)
- Alta dimensionalidad de la entrada (mayor a 10)
- Cantidad de parámetros a entrenar mayor (o al menos comparable) que la cantidad de muestras.
- Suficientes hiperparámetros para considerar hacer un barrido en grilla una mala opción.

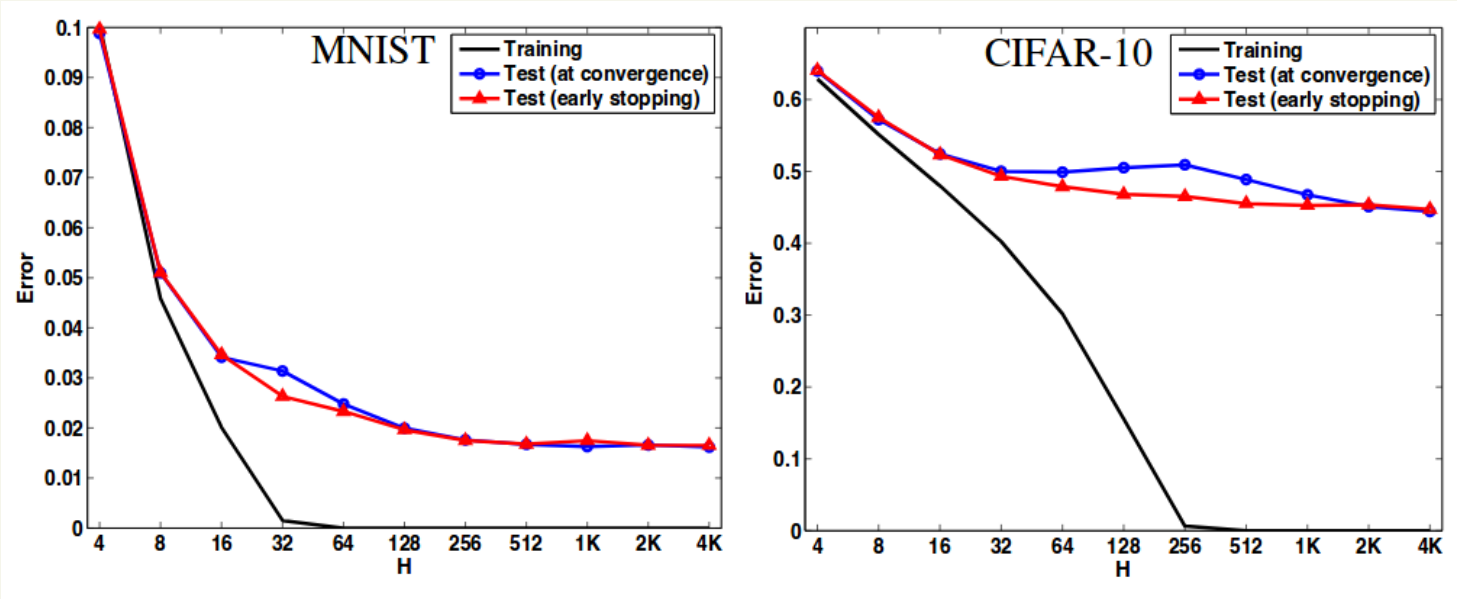
Compromiso Sesgo/Varianza

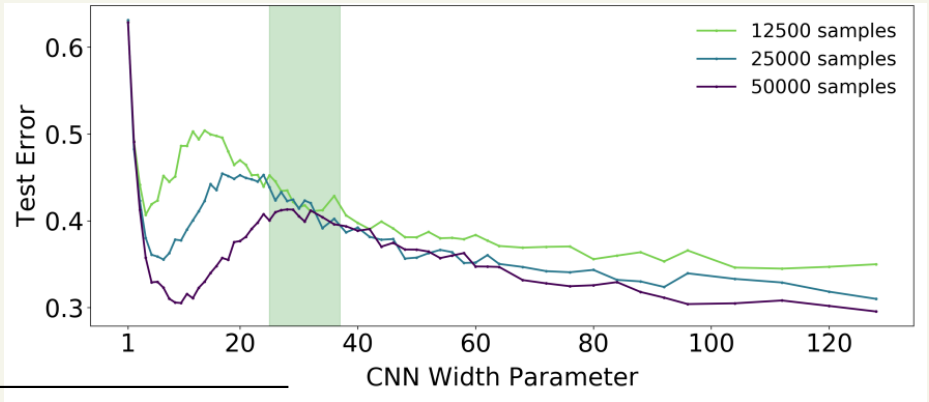
Aprendizaje Clásico



Compromiso Sesgo/Varianza

Deep Learning





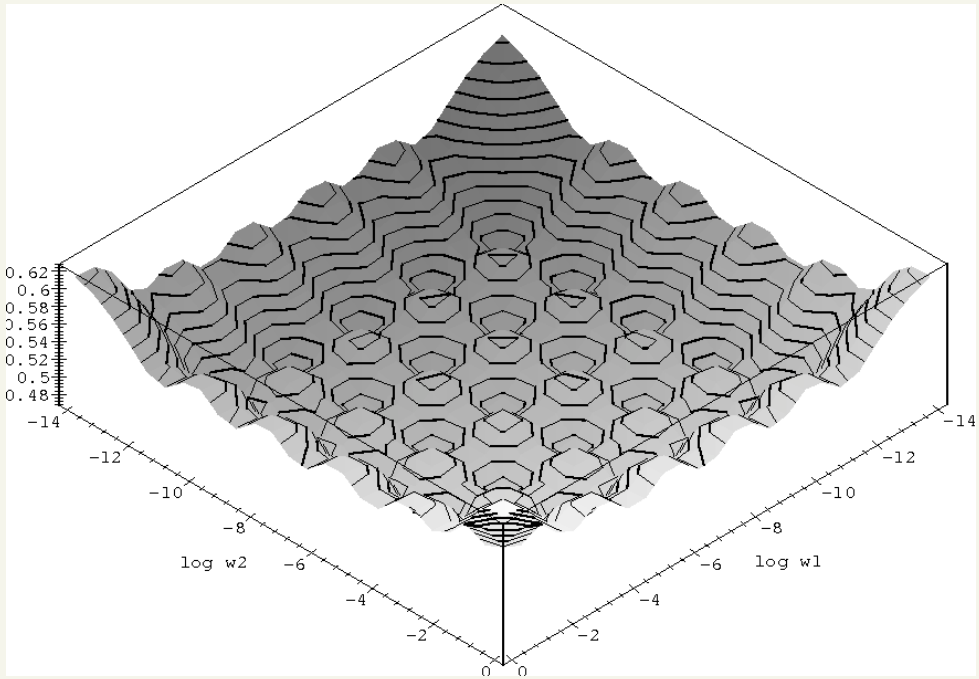
Neyshabur et al. 2017: "Geometry of optimization and implicit regularization in deep learning".

Nakkiran et al. 2019: "Deep Double Descent: Where bigger models and more data hurt".

La sobreparametrización

Otros impactos de la sobreparametrización:

- Muchos *saddle points*.
- La cantidad de mínimos locales es exponencial con la dimensión.
- Incluso los mínimos globales no son todos iguales para la generalización.

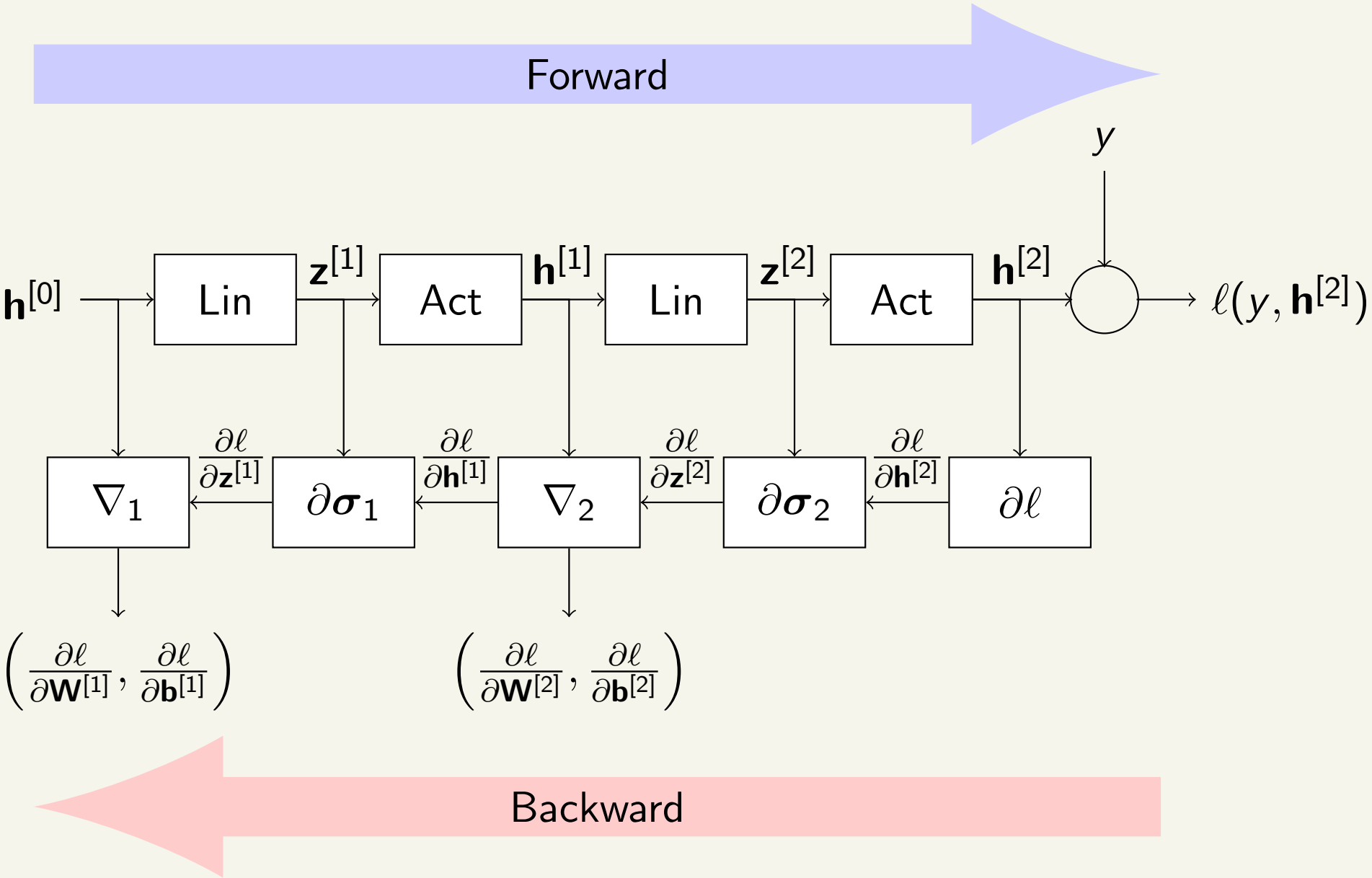


Auer et al. 1995: "Exponentially many local minima for single neurons".

Agenda

1. Redes Neuronales y el Deep World
2. Entrenamiento de Redes Neuronales Profundas
3. Detalles de Deep Learning
4. Regularización
5. Redes Convolucionales
6. Redes Recurrentes

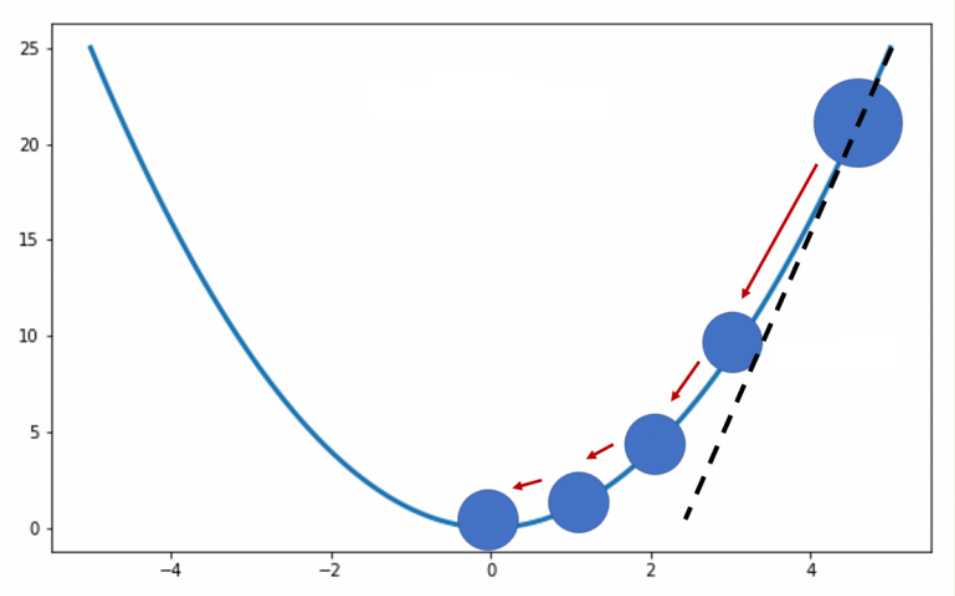
Forward-Backward Propagation



Gradiente Descendente

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{n_{\text{tr}}} \sum_{i=1}^{n_{\text{tr}}} \frac{\partial \ell(y_i, \mathbf{h}_i^{[L]})}{\partial \theta} = \mathbf{0}$$

$$\theta_{t+1} = \theta_t - \alpha \frac{\partial J(\theta_t)}{\partial \theta}$$



Si tengo muchos datos y muchos parámetros es muy pesado

Gradiente Descendente Estocástico

Pero computar n_{tr} gradientes en cada iteración es pesado, entonces surge la versión estocástica:

$$W_{t+1} = W_t - \alpha \left. \frac{\partial L(X_t, Y_t; w)}{\partial w} \right|_{w=W_t}$$

donde en cada paso se sortea una nueva variable aleatoria (es decir muestras distintas) introduciendo una aleatoriedad al problema. Algunas características son:

- El cómputo es mucho mas liviano que la versión clásica.
- La aleatoriedad me hace converger en una "bola de ruido" (problema de sesgo).
- Mejora la capacidad de generalización agregando ruido al proceso de aprendizaje.

Análisis de convergencia

Supuestos

- Los pares $\{(X_i, Y_i)\}_{i \in \mathbb{N}} \sim (X, Y)$ son i.i.d. e independientes de W_0.
- Los gradientes tienen varianza acotada:

$$\text{var}\left(\frac{\partial L(X, Y; w)}{\partial w}\right) \leq \sigma^2 \qquad \forall\, w \in \Theta$$

- La magnitud $\mathcal{L}(w) = \mathbb{E}[L(X, Y; w)]$ posee un gradiente acotado como $\|\nabla \mathcal{L}(w)\|^2 \leq \mu^2$ para todo $w \in \Theta$.
- $\mathcal{L}(w)$ es fuertemente convexa. Es decir que existe $c > 0$ tal que

$$\mathcal{L}(w_1) \geq \mathcal{L}(w_2) + \langle \nabla \mathcal{L}(w_2); w_1 - w_2 \rangle + \frac{c}{2}\|w_1 - w_2\|^2$$

  para todo w_1 y w_2.
- Tecnicismos (existencia de las derivadas, momentos e intercambio).

G. Turinici: "The convergence of the Stochastic Gradient Descent (SGD) : a self-contained proof", 2021.

Análisis de convergencia

Solución óptima

La condición de convexidad asegura que existe un único w^* tal que $\mathcal{L}(w^*) \leq \mathcal{L}(w)$ para todo $w \in \Theta$ y por lo tanto $\nabla\mathcal{L}(w^*) = 0$. La convexidad fuerte asegura entonces que:

$$\mathcal{L}(w) \geq \mathcal{L}(w^*) + \frac{c}{2}\|w - w^*\|^2, \qquad \forall\, w \in \Theta$$

Análisis de convergencia

Solución óptima

La condición de convexidad asegura que existe un único w^* tal que $\mathcal{L}(w^*) \leq \mathcal{L}(w)$ para todo $w \in \Theta$ y por lo tanto $\nabla\mathcal{L}(w^*) = 0$. La convexidad fuerte asegura entonces que:

$$\mathcal{L}(w) \geq \mathcal{L}(w^*) + \frac{c}{2}\|w - w^*\|^2, \qquad \forall\, w \in \Theta$$

$$\mathbb{E}\left[\|W_{t+1} - w^*\|^2\right] = \mathbb{E}\left[\left\|W_t - w^* - \alpha \left.\frac{\partial L(X_t, Y_t; w)}{\partial w}\right|_{w=W_t}\right\|^2\right]$$

$$= \mathbb{E}\left[\|W_t - w^*\|^2\right] + \alpha^2\mathbb{E}\left[\left\|\left.\frac{\partial L(X_t, Y_t; w)}{\partial w}\right|_{w=W_t}\right\|^2\right]$$

$$- 2\alpha\mathbb{E}\left[\left\langle \left.\frac{\partial L(X_t, Y_t; w)}{\partial w}\right|_{w=W_t}; W_t - w^* \right\rangle\right]$$

Análisis de convergencia

Independencia

Como $W_t = f\left(W_0, \{(X_k, Y_k)_{k=1}^{t-1}\}\right)$, es independiente del par (X_t, Y_t).

Análisis de convergencia

Independencia

Como $W_t = f\left(W_0, \{(X_k, Y_k)_{k=1}^{t-1}\}\right)$, es independiente del par (X_t, Y_t).

$$
\begin{aligned}
&\mathbb{E}\left[\left\langle \frac{\partial L(X_t, Y_t; w)}{\partial w}\bigg|_{w=W_t} ; W_t - w^* \right\rangle \middle| W_t = w\right] \\
&= \mathbb{E}\left[\left\langle \frac{\partial L(X_t, Y_t; w)}{\partial w} ; w - w^* \right\rangle\right] \\
&= \langle \nabla \mathcal{L}(w); w - w^* \rangle \\
&\geq \mathcal{L}(w) - \mathcal{L}(w^*) + \frac{c}{2}\|w - w^*\|^2 \\
&\geq c\|w - w^*\|^2
\end{aligned}
$$

Análisis de convergencia

$$\mathbb{E}\left[\left\|\frac{\partial L(X_t, Y_t; w)}{\partial w}\Big|_{w=W_t}\right\|^2 \middle| W_t = w\right] = \mathbb{E}\left[\left\|\frac{\partial L(X, Y; w)}{\partial w}\right\|^2\right]$$

$$= \text{var}\left(\frac{\partial L(X, Y; w)}{\partial w}\right) + \left\|\mathbb{E}\left[\frac{\partial L(X, Y; w)}{\partial w}\right]\right\|^2 \leq \sigma^2 + \mu^2$$

Análisis de convergencia

$$\mathbb{E}\left[\left\|\frac{\partial L(X_t, Y_t; w)}{\partial w}\Big|_{w=W_t}\right\|^2 \middle| W_t = w\right] = \mathbb{E}\left[\left\|\frac{\partial L(X, Y; w)}{\partial w}\right\|^2\right]$$

$$= \text{var}\left(\frac{\partial L(X, Y; w)}{\partial w}\right) + \left\|\mathbb{E}\left[\frac{\partial L(X, Y; w)}{\partial w}\right]\right\|^2 \leq \sigma^2 + \mu^2$$

$$\begin{aligned}
&\mathbb{E}\left[\|W_{t+1} - w^*\|^2\right] \\
&\quad \leq \mathbb{E}\left[\|W_t - w^*\|^2\right] + \alpha^2(\sigma^2 + \mu^2) - 2\alpha c\mathbb{E}\left[\|W_t - w^*\|^2\right] \\
&\quad = (1 - 2\alpha c)\,\mathbb{E}\left[\|W_t - w^*\|^2\right] + \alpha^2(\sigma^2 + \mu^2)
\end{aligned}$$

Análisis de convergencia

$$\mathbb{E}\left[\left\|\frac{\partial L(X_t, Y_t; w)}{\partial w}\Big|_{w=W_t}\right\|^2 \Big| W_t = w\right] = \mathbb{E}\left[\left\|\frac{\partial L(X, Y; w)}{\partial w}\right\|^2\right]$$

$$= \text{var}\left(\frac{\partial L(X, Y; w)}{\partial w}\right) + \left\|\mathbb{E}\left[\frac{\partial L(X, Y; w)}{\partial w}\right]\right\|^2 \leq \sigma^2 + \mu^2$$

$$\begin{aligned}
&\mathbb{E}\left[\|W_{t+1} - w^*\|^2\right] \\
&\quad \leq \mathbb{E}\left[\|W_t - w^*\|^2\right] + \alpha^2(\sigma^2 + \mu^2) - 2\alpha c\mathbb{E}\left[\|W_t - w^*\|^2\right] \\
&\quad = (1 - 2\alpha c)\,\mathbb{E}\left[\|W_t - w^*\|^2\right] + \alpha^2(\sigma^2 + \mu^2)
\end{aligned}$$

Bola de ruido

$$\limsup_{t\to\infty} \mathbb{E}\left[\|W_t - w^*\|^2\right] \leq \alpha \cdot \frac{\sigma^2 + \mu^2}{2c}$$

Tradeoff entre bola de ruido y velocidad de convergencia!

Gradiente Descendente Estocástico por minibatch

¿No habrá nada intermedio?

$$W_{t+1} = W_t - \alpha \left. \frac{1}{B} \sum_{i=1}^{B} \frac{\partial L\left(X_t^{(i)}, Y_t^{(i)}, w\right)}{\partial w} \right|_{W_t = w}$$

donde en cada paso se sortean B una nuevas variables aleatorias i.i.d. El "batchsize" B nos permite explotar el tradeoff según nuestra conveniencia. En el análisis de la bola de ruido, la función $\mathcal{L}(w)$ es la misma, lo único que cambia es σ^2:

$$\operatorname{var}\left(\frac{\partial \frac{1}{B} \sum_{i=1}^{B} L\left(X^{(i)}, Y^{(i)}; w\right)}{\partial w} \right) \leq \frac{\sigma^2}{B}$$

Obteniendo una bola de ruido de la forma:

$$\limsup_{t \to \infty} \mathbb{E}\left[\| W_t - w^* \|^2 \right] \leq \alpha \cdot \frac{\frac{\sigma^2}{B} + \mu^2}{2c}$$

Sobre el batchsize

- Batchs grandes proporcionan una estimación más precisa del gradiente, pero de forma mas pesada a nivel computacional.
- Las arquitecturas multicore poseen un mínimo valor del batchsize debajo del cual no hay reducción en el tiempo de procesamiento.
- Si todos las muestras del batch se procesan en paralelo, entonces la cantidad de memoria escala con el tamaño del batch, limitando superiormente el batchsize.
- Algunos tipos de hardware logran un mejor tiempo de ejecución con tamaños específicos de arrays. Especialmente las GPU suelen ser más rápidas con batchsize que sean potencias de 2.
- Los batchs pequeños ofrecen un efecto de regularización debido al ruido que se agrega al proceso de aprendizaje. La generalización suele ser mejor para un batchsize de 1. Pero batchs tan pequeños requieren bajos learning rates lo que puede demorar el proceso de aprendizaje.

Goodfellow, Bengio and Courville: "Deep Learning", chapter 8.

Sobre el batchsize

En la práctica...

Se utilizan datos distintos hasta que se acaban, realizando en principio n_{tr}/B pasos del algoritmo. Como esto suele ser insuficiente, los datos se mezclan y se vuelven a usar (construyendo batches distintos). A cada pasada de todos los datos se lo llama epoch.

Hiperparámetros de esta etapa:

- Batch size (B)
- Número de epochs (o condición de stop)

Diferentes tipos de optimizadores

Algunas variantes del gradiente descendente estocástico (SGD):

- SGD with momentum
- Nesterov
- RMSProp
- Adagrad
- Adam

Momentum: Exponentially Weighted Averages

Objetivo

Bajar la bola de ruido con alta velocidad de aprendizaje (α grande) sin aumentar la carga computacional (batchsize moderado). Es decir, buscar reemplazos para $\mathbf{g}_t = \frac{1}{B}\sum_{i=1}^{B} \frac{\partial L_i(\theta_{t-1})}{\partial \theta}$.

Momentum (no le creas tanto): $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t$ con $\beta_1 \in [0, 1)$ (típico $\beta_1 = 0.9$).

Momentum: Exponentially Weighted Averages

Objetivo

Bajar la bola de ruido con alta velocidad de aprendizaje (α grande) sin aumentar la carga computacional (batchsize moderado). Es decir, buscar reemplazos para $\mathbf{g}_t = \frac{1}{B}\sum_{i=1}^{B} \frac{\partial L_i(\theta_{t-1})}{\partial \theta}$.

Momentum (no le creas tanto): $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t$ con $\beta_1 \in [0, 1)$ (típico $\beta_1 = 0.9$).

$$\mathbf{m}_5 = (1-\beta_1)\mathbf{g}_5 + (1-\beta_1)\beta_1\mathbf{g}_4 + (1-\beta_1)\beta_1^2\mathbf{g}_3 + (1-\beta_1)\beta_1^3\mathbf{g}_2 + (1-\beta_1)\beta_1^4\mathbf{g}_1$$

Momentum: Exponentially Weighted Averages

Objetivo

Bajar la bola de ruido con alta velocidad de aprendizaje (α grande) sin aumentar la carga computacional (batchsize moderado). Es decir, buscar reemplazos para $\mathbf{g}_t = \frac{1}{B}\sum_{i=1}^{B} \frac{\partial L_i(\theta_{t-1})}{\partial \theta}$.

Momentum (no le creas tanto): $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t$ con $\beta_1 \in [0,1)$ (típico $\beta_1 = 0.9$).

$$\mathbf{m}_5 = (1-\beta_1)\mathbf{g}_5 + (1-\beta_1)\beta_1\mathbf{g}_4 + (1-\beta_1)\beta_1^2\mathbf{g}_3 + (1-\beta_1)\beta_1^3\mathbf{g}_2 + (1-\beta_1)\beta_1^4\mathbf{g}_1$$

$$\mathbf{m}_t = (1-\beta_1)\sum_{k=0}^{t-1}\beta_1^k \mathbf{g}_{t-k}$$

Respuesta: $\theta_t = \theta_{t-1} - \alpha\mathbf{m}_t$.
Learning rate efectivo: Fue atenuado $\alpha_{\text{ef}} = \alpha(1-\beta_1)$.

$$\theta_t = \theta_{t-1} - \alpha\beta_1\mathbf{m}_{t-1} - \alpha(1-\beta_1)\mathbf{g}_t$$

Bias Correction

Momentum subestima mucho los gradientes en las primeras iteraciones:

$$\mathbf{m}_1 = (1 - \beta_1)\mathbf{g}_1 = 0.1\mathbf{g}_1$$
$$\mathbf{m}_2 = (1 - \beta_1)\mathbf{g}_2 + (1 - \beta_1)\beta_1\mathbf{g}_1 = 0.1\mathbf{g}_2 + 0.09\mathbf{g}_1$$

Bias Correction

Momentum subestima mucho los gradientes en las primeras iteraciones:

$$\mathbf{m}_1 = (1-\beta_1)\mathbf{g}_1 = 0.1\mathbf{g}_1$$
$$\mathbf{m}_2 = (1-\beta_1)\mathbf{g}_2 + (1-\beta_1)\beta_1\mathbf{g}_1 = 0.1\mathbf{g}_2 + 0.09\mathbf{g}_1$$

Solución: Verifiquemos que sea una combinación convexa de gradientes.

$$\hat{\mathbf{m}}_t = \frac{(1-\beta_1)\sum_{k=0}^{t-1}\beta_1^k\mathbf{g}_{t-k}}{(1-\beta_1)\sum_{k=0}^{t-1}\beta_1^k} = \frac{\mathbf{m}_t}{1-\beta_1^t}$$

Respuesta: $\theta_t = \theta_{t-1} - \alpha\hat{\mathbf{m}}_t$.
Learning rate efectivo: Atenuación paulatina $\alpha_{\text{ef}} = \alpha\frac{1-\beta_1}{1-\beta_1^t}$.

Nesterov's accelerated gradient

Classical momentum: $\theta_t = \theta_{t-1} - \alpha\beta_1 \mathbf{m}_{t-1} - \alpha(1-\beta_1)\frac{\partial L(\theta_{t-1})}{\partial \theta}$

Sutskever et al. 2013: "On the importance of initialization and momentum in deep learning".

Nesterov's accelerated gradient

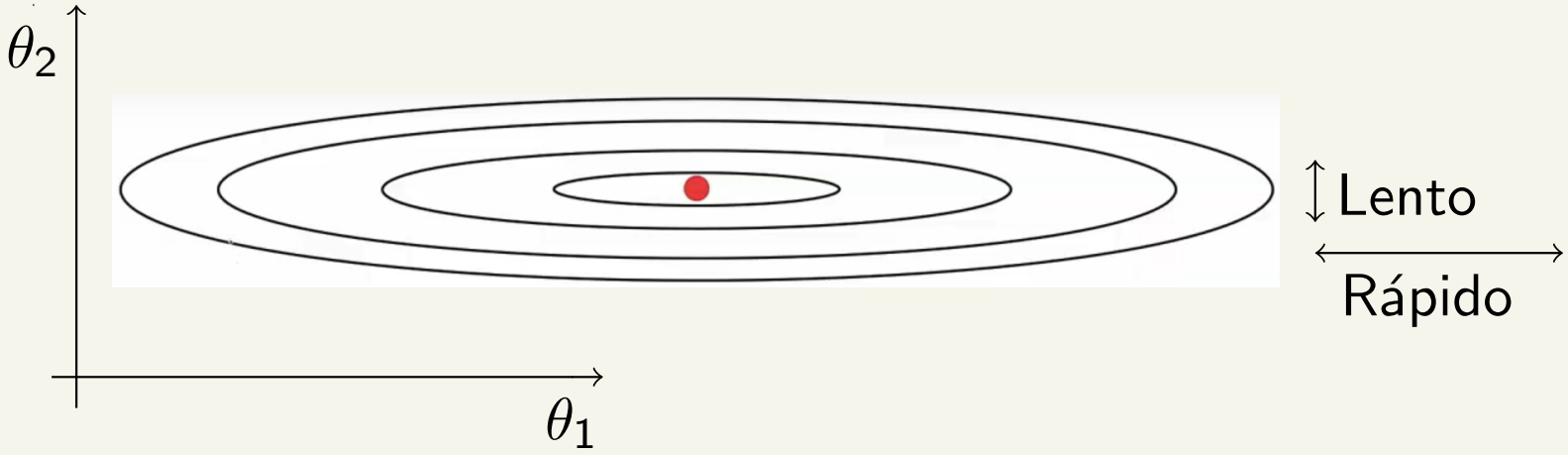
Classical momentum: $\theta_t = \theta_{t-1} - \alpha\beta_1 \mathbf{m}_{t-1} - \alpha(1-\beta_1)\frac{\partial L(\theta_{t-1})}{\partial \theta}$

Nesterov momentum: $\theta_t = \theta_{t-1} - \alpha\beta_1 \mathbf{m}_{t-1} - \alpha(1-\beta_1)\frac{\partial L(\theta_{t-1} - \alpha\beta_1 \mathbf{m}_{t-1})}{\partial \theta}$

La idea del gradiente es que me diga cuanto y hacia donde me tengo que mover para acercarme al mínimo. Nesterov usa toda la información que tengo actualmente antes de elegir cuanto y hacia donde moverse.

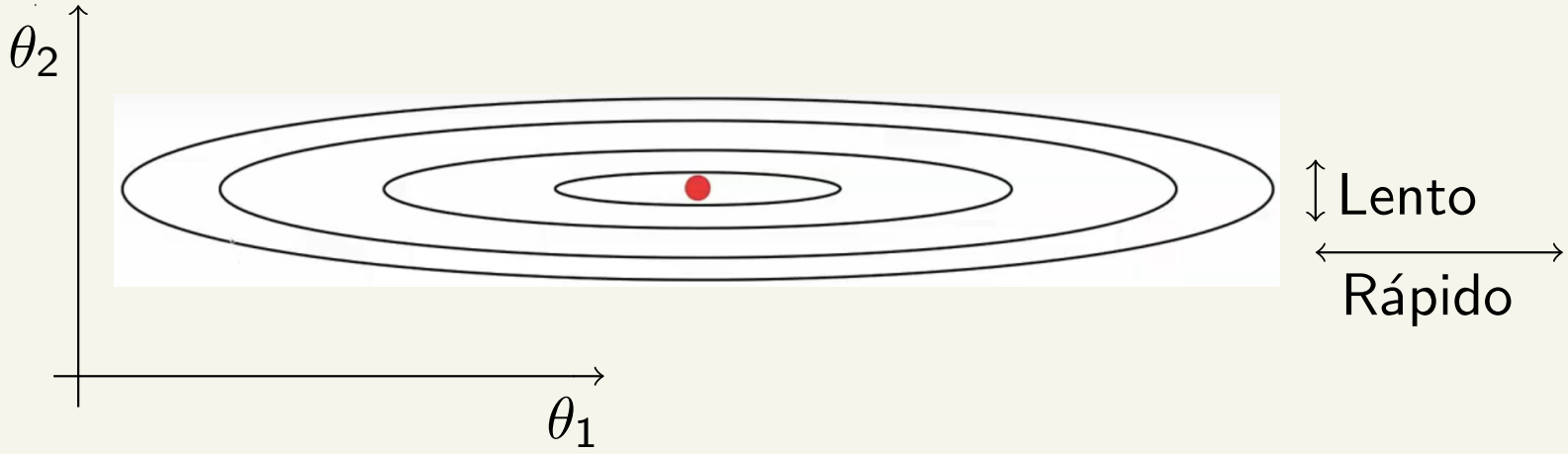
Sutskever et al. 2013: "On the importance of initialization and momentum in deep learning".

Adaptive Gradient (Adagrad)



Duchi et al. 2011: "Adaptive subgradient methods for online learning and stochastic optimization".

Adaptive Gradient (Adagrad)



Solución: Usar la historia del gradiente para escalarlo. Se define un parámetro que evite errores numéricos $\epsilon \sim 10^{-8}$.

Respuesta: $\theta_t = \theta_{t-1} - \alpha \frac{\mathbf{g}_t}{\sqrt{\sum_{\tau=1}^{t} \mathbf{g}_\tau^2 + \epsilon}}$.

Learning rate efectivo: Se adapta a cada componente.

Duchi et al. 2011: "Adaptive subgradient methods for online learning and stochastic optimization".

Root Mean Square Propagation

RMSProp

Exponentially weighted averages para los cuadrados: $\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2$.
Respuesta: $\theta_t = \theta_{t-1} - \alpha \frac{\mathbf{g}_t}{\sqrt{\mathbf{v}_t} + \epsilon}$.

RMSProp with bias correction

Misma idea: $\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t}$.
Respuesta: $\theta_t = \theta_{t-1} - \alpha \frac{\mathbf{g}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}$.

ADAM

Combinar RMSProp con momentum (ambos con bias correction).
Respuesta: $\theta_t = \theta_{t-1} - \alpha \frac{\mathbf{m}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}$.

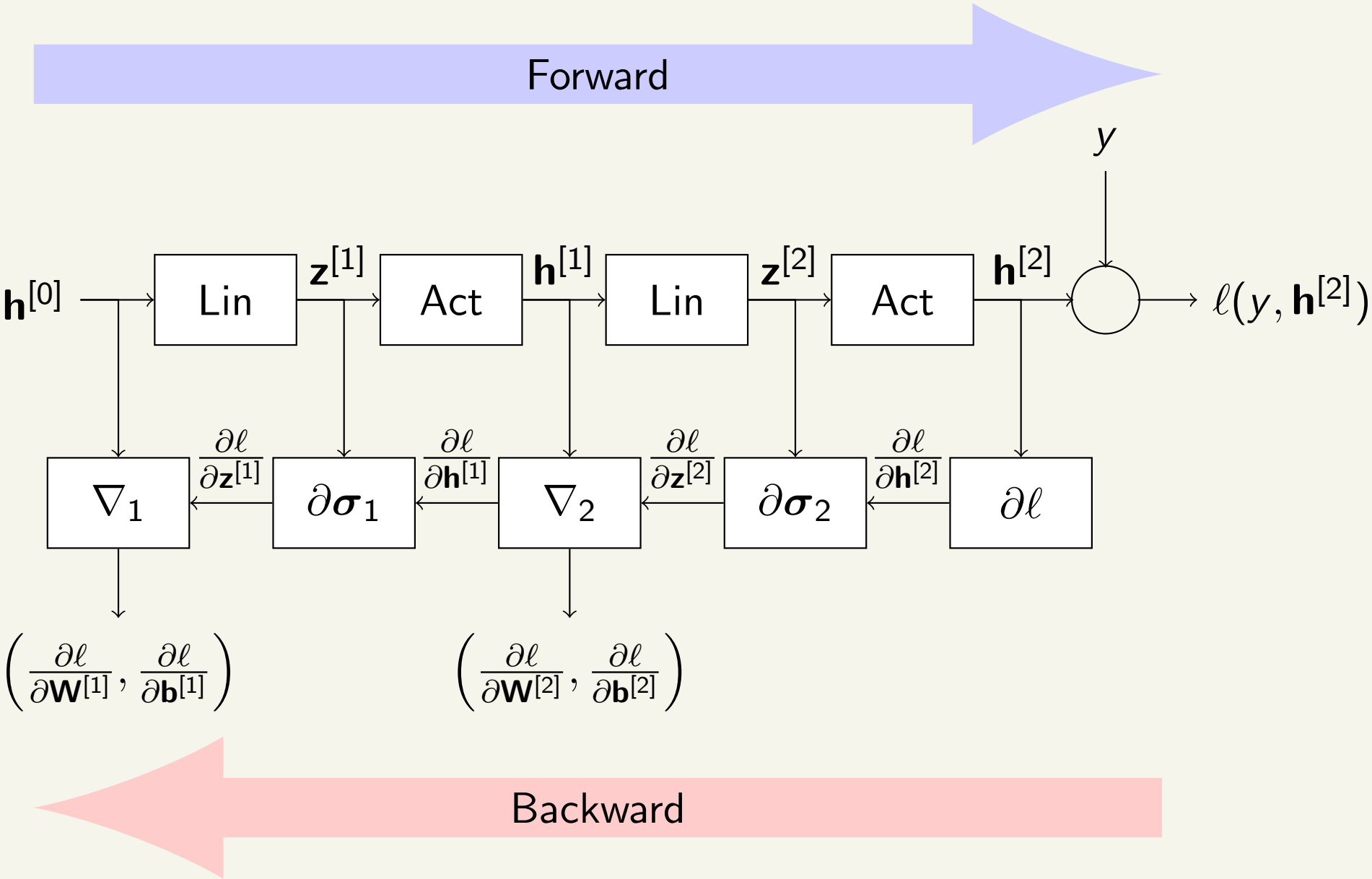
Kingma-Lei Ba 2015: "ADAM: A method for stochastic optimization".

Learning rate decay

Puedo ir reduciendo la bola de ruido a medida que voy aprendiendo. Sea $e_t = 1 + \lfloor \frac{tB}{n_{\text{tr}}} \rfloor$ el número de epoch que estoy en la iteración t, luego algunas de las expresiones más utilizadas son:

- $\alpha = \alpha_0 \frac{1+\delta}{1+\delta e_t}$
- $\alpha = \alpha_0 \gamma^{e_t - 1}$
- $\alpha = \frac{\alpha_0}{\sqrt{e_t}}$
- Definir α constante por intervalos de e_t (de forma decreciente)

Forward-Backward Propagation



Agenda

1. Redes Neuronales y el Deep World
2. Entrenamiento de Redes Neuronales Profundas
3. Detalles de Deep Learning
4. Regularización
5. Redes Convolucionales
6. Redes Recurrentes

Vanishing/Exploting

Idea

Para redes muy profundas, la composición de layers puede generar magnitudes muy pequeñas o muy grandes que estropee la precisión numérica. Este efecto puede ocurrir tanto en las magnitudes (forward) o en sus gradientes (backward).

Comparación sigmoide vs ReLU

$\sigma(z) = \frac{1}{1+e^{-z}}$

- No explota $\sigma(z) < 1$
- No explota su gradiente $\sigma'(z) = \sigma(z)\,(1 - \sigma(z)) \leq \frac{1}{4}$
- Su gradiente se desvanece con mucha facilidad

$\sigma(z) = \max\{0, z\}$

- Su gradiente no desvancece ni explota $\sigma'(z) = \mathbb{1}\{z > 0\}$.
- Tiene tendencia a explotar $\sigma(z) \geq z$
- Es computacionalmente más simple de computar
- Puede matar neuronas: $\left(\mathbf{h}^{[l]}\right)_k = 0$ para todo $\mathbf{h}^{[l-1]}$ (cambiarla por Leaky-Relu)

¿Como evitar la explosión de la ReLU?

Una manera de atenuar una posible explosión es con una inicialización adecuada. Sea $\mathbf{b} = \mathbf{0}$ y que $w_k \sim \mathcal{N}(0, v)$ i.i.d. Tomemos una neurona: $u = \sum_{k=1}^{d} w_k \sigma(z_k)$ y supongamos que todos los z_k son identicamente distribuidos con una densidad simétrica en el origen independiente de la inicialización $w_1, \cdots, w_d$.

¿Como evitar la explosión de la ReLU?

Una manera de atenuar una posible explosión es con una inicialización adecuada. Sea $\mathbf{b} = \mathbf{0}$ y que $w_k \sim \mathcal{N}(0, v)$ i.i.d. Tomemos una neurona: $u = \sum_{k=1}^{d} w_k \sigma(z_k)$ y supongamos que todos los z_k son identicamente distribuidos con una densidad simétrica en el origen independiente de la inicialización $w_1, \cdots, w_d$.

$$
\begin{aligned}
\operatorname{var}(u) &= \mathbb{E}\left[u^2\right] \\
&= \sum_{i=1}^{d} \sum_{j=1}^{d} \mathbb{E}[w_i w_j] \mathbb{E}\left[\sigma(z_i)\, \sigma(z_j)\right] \\
&= \sum_{k=1}^{d} v \mathbb{E}\left[\sigma^2(z_k)\right] \\
&= dv \mathbb{E}\left[z_1^2 \mathbb{1}\{z_1 > 0\}\right] \\
&= \frac{dv}{2} \mathbb{E}\left[z_1^2\right]
\end{aligned}
$$

Inicialización de parámetros

$$\mathbb{E}\left[\left(z^{[l]}\right)^2\right] = \frac{d^{[l-1]}v}{2}\mathbb{E}\left[\left(z^{[l-1]}\right)^2\right]$$

Para mantener la variabilidad se propone $(\mathbf{w}^{[l]})_{i,j} \sim \mathcal{N}\left(0, \frac{2}{d^{[l-1]}}\right)$ para matrices que multiplican salidas de ReLU. (He Normal Initialization)
Otras inicializaciones:

- $(\mathbf{w}^{[l]})_{i,j} \sim \mathcal{N}\left(0, \frac{1}{d^{[l-1]}}\right)$ (Lecun Normal Initialization)
- $(\mathbf{w}^{[l]})_{i,j} \sim \mathcal{N}\left(0, \frac{2}{d^{[l-1]}+d^{[l]}}\right)$ (Glorot Normal Initialization)
- $(\mathbf{w}^{[l]})_{i,j} \sim \mathcal{U}\left(-\sqrt{\frac{6}{d^{[l-1]}+d^{[l]}}}, \sqrt{\frac{6}{n^{[l-1]}+n^{[l]}}}\right)$ (Glorot Uniform Initialization)
- Normales truncadas (si excede 2 desvíos va de nuevo)

He et al. 2015: "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification".

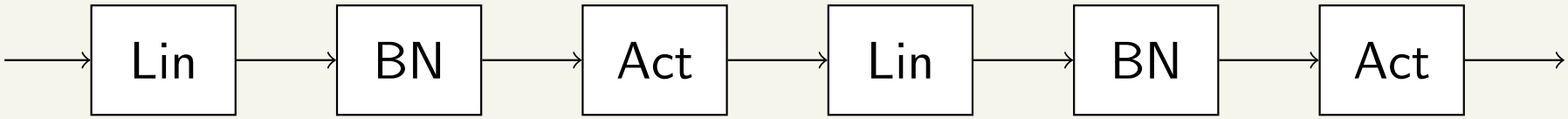
Glorot-Bengio 2010: "Understanding the difficulty of training deep feedforward neural networks"

Batch Normalization

Batch Normalization es la generalización de input normalization. Algunas de sus ventajas son:

- Acelera el proceso de aprendizaje
- Al normalizar por batch agrega cierto ruido en el proceso de aprendizaje lo que puede ayudar a la generalización.
- Colabora en reducir el vanishing y exploting.
- Otorga cierta robustez a cambios en la distribución de la entrada (?).

Usualmente se aplica sobre la salida del layer lineal, previo a las activaciones. El último layer no suele llevar batch normalization.



Ioffe-Szegedy 2015: "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift".

Batch Normalization

Magnitudes locales de cada batch

Sea $x \in \mathbb{R}^{n \times L \times d}$, en cada paso del SGD se calcula:

$$\mu_j = \frac{1}{nL} \sum_{t=1}^{L} \sum_{i=1}^{n} x_{i,t,j}, \qquad \sigma_j^2 = \frac{1}{nL} \sum_{t=1}^{L} \sum_{i=1}^{n} (x_{i,t,j} - \mu_j)^2$$

Las magnitudes μ_j y σ_j^2 se definen localmente por cada batch.

Batch Normalization

Magnitudes locales de cada batch

Sea $x \in \mathbb{R}^{n \times L \times d}$, en cada paso del SGD se calcula:

$$\mu_j = \frac{1}{nL} \sum_{t=1}^{L} \sum_{i=1}^{n} x_{i,t,j}, \qquad \sigma_j^2 = \frac{1}{nL} \sum_{t=1}^{L} \sum_{i=1}^{n} (x_{i,t,j} - \mu_j)^2$$

Las magnitudes μ_j y σ_j^2 se definen localmente por cada batch.

Parámetros entrenables

La salida del batch normalization se computa como:

$$\text{BN}(x)_{i,t,j} = \frac{x_{i,t,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}} \cdot \gamma_j + \beta_j$$

donde γ_j y β_j se entrenan con el SGD.

Algunas consideraciones sobre batch normalization

Parámetros no entrenables

Al momento de hacer inferencia se analiza por muestra, por lo que es necesario estimar (durante el entrenamiento) $\hat{\mu}_j$ y $\hat{\sigma}_j^2$. Normalmente esto se hace con Exponentially Weighted Averages (iterativo):

$$\hat{\mu}_j^{[k]} = \eta_\mu \cdot \hat{\mu}_j^{[k-1]} + (1 - \eta_\mu) \cdot \mu_j, \quad \hat{\sigma}_j^{2[k]} = \eta_\sigma \cdot \hat{\sigma}_j^{2[k-1]} + (1 - \eta_\sigma) \cdot \sigma_j^2$$

Los parámetros $\hat{\mu}_j$ y $\hat{\sigma}_j^2$ se llaman no entrenables porque no se usan durante el entrenamiento.

Algunas consideraciones sobre batch normalization

Parámetros no entrenables

Al momento de hacer inferencia se analiza por muestra, por lo que es necesario estimar (durante el entrenamiento) $\hat{\mu}_j$ y $\hat{\sigma}_j^2$. Normalmente esto se hace con Exponentially Weighted Averages (iterativo):

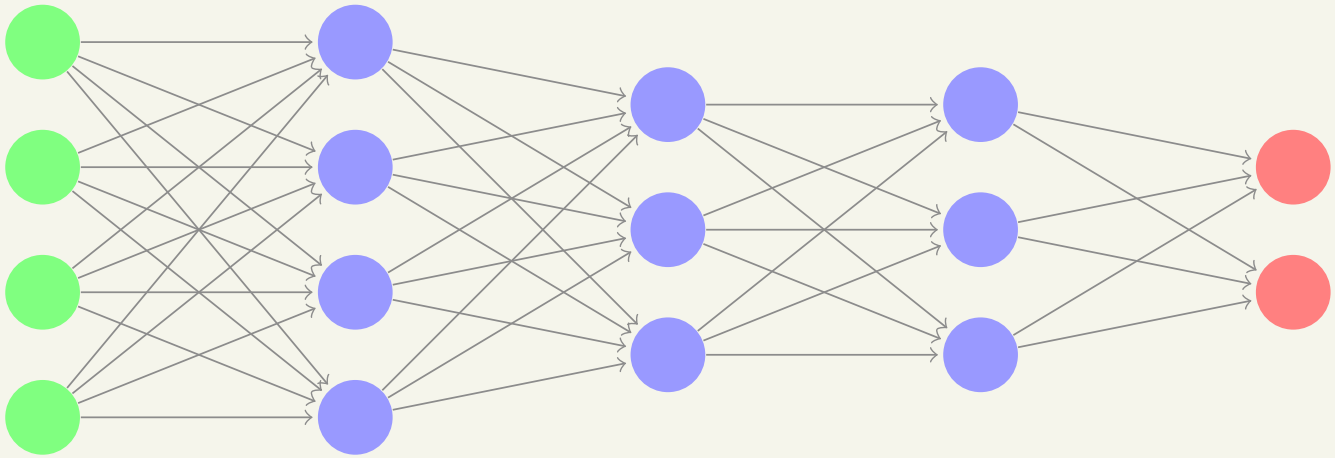
$$\hat{\mu}_j^{[k]} = \eta_\mu \cdot \hat{\mu}_j^{[k-1]} + (1 - \eta_\mu) \cdot \mu_j, \quad \hat{\sigma}_j^{2[k]} = \eta_\sigma \cdot \hat{\sigma}_j^{2[k-1]} + (1 - \eta_\sigma) \cdot \sigma_j^2$$

Los parámetros $\hat{\mu}_j$ y $\hat{\sigma}_j^2$ se llaman no entrenables porque no se usan durante el entrenamiento.

- La única función de ϵ es evitar dividir por números muy chicos.
- Dada la naturaleza ruidosa de μ_j y σ_j^2, se puede pensar batch normalization como una inyección de ruido en las unidades ocultas.
- Esta normalización se hace posterior a layers lineales, por lo que en dichos layers deja de ser necesario el bias, porque se acapara en β_j.
- Un layer batchnorm agrega d parámetros entrenables (si se saca el bias) y $2d$ no entrenables.

Covariate shift

Cambia p_X pero no $P_{Y|X}$

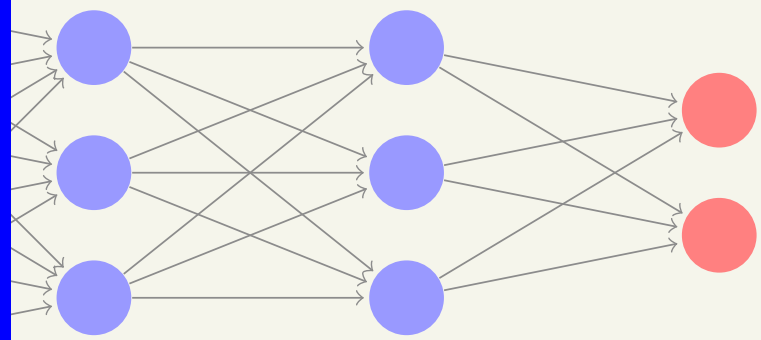


Santurkar et al. 2019: "How Does Batch Normalization Help Optimization?".

Covariate shift

Cambia p_X pero no $P_{Y|X}$

Para cada layer, los layer anteriores son una co-variate shif permanente



Santurkar et al. 2019: "How Does Batch Normalization Help Optimization?".

Layer Normalization

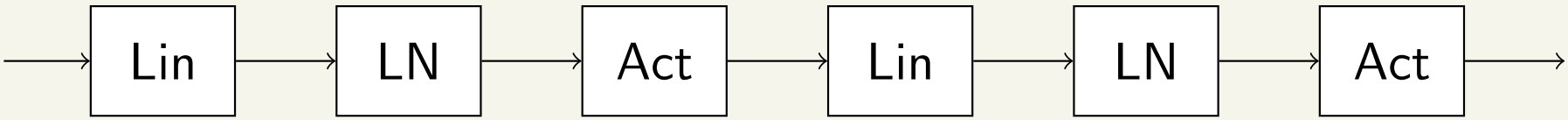
Layer Normalization

Normalizar vectores latentes para estabilizar la escala y hacer el entrenamiento más robusto.

$$\mathrm{LN}(x)_{i,t,j} = \frac{x_{i,t,j} - \mu_{i,t}}{\sigma_{i,t} + \epsilon} \cdot \gamma_j + \beta_j$$

donde ϵ es una constante que evita dividir por cero. Si $x \in \mathbb{R}^{n \times L \times d}$, $\gamma, \beta \in \mathbb{R}^d$ son parámetros a aprender y $\mu, \sigma \in \mathbb{R}^{n \times L}$ se definen en como:

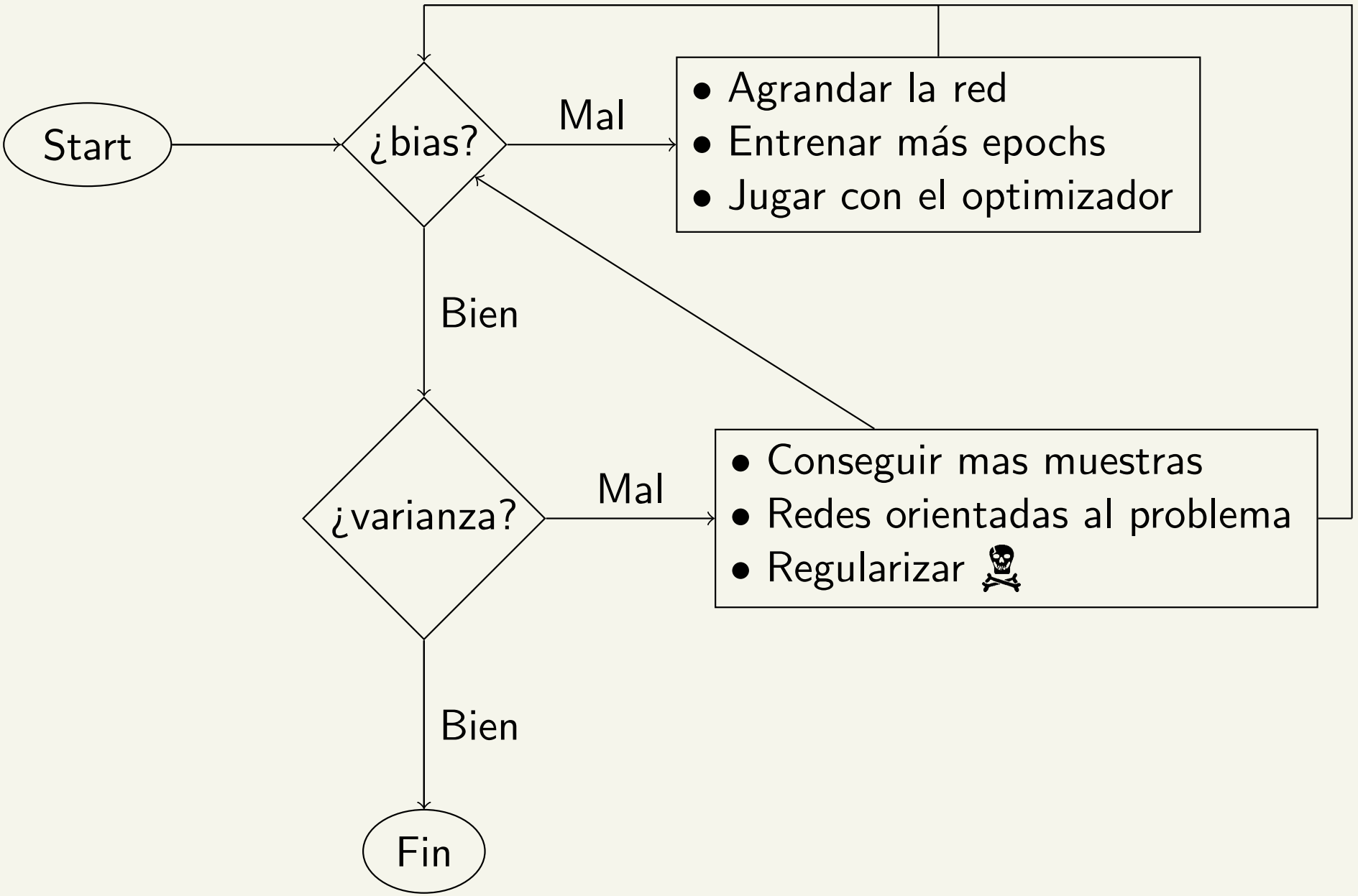
$$\mu_{i,t} = \frac{1}{d} \sum_{j=1}^{d} x_{i,t,j}, \qquad \sigma_{i,t} = \sqrt{\frac{1}{d} \sum_{j=1}^{d} (x_{i,t,j} - \mu_{i,t})^2}$$



Agenda

1. Redes Neuronales y el Deep World
2. Entrenamiento de Redes Neuronales Profundas
3. Detalles de Deep Learning
4. Regularización
5. Redes Convolucionales
6. Redes Recurrentes

Propuesta de algortimo (primeras iteraciones)



Atacar el punto débil (refinamiento)

¿Que conviene corregir? ¿Sesgo o varianza?

- **Avoidable bias**: Error de train - Error bayesiano
- **Generalization Gap**: Error de validación - Error de train

Ambos problemas se atacan por separado!

Tipos de regularizaciones (algunos)

¿Cómo mejorar la capacidad de generalización del algoritmo?

- **Término de penalización (o regularizador)**: Incluir un término adicional en la función costo.
- **Inyección de ruido**: Incluir algún tipo de ruido en el proceso de aprendizaje.
- **Early stopping (parado temprano)**: Detener el proceso de aprendizaje tempranamente con algún criterio razonable.
- **Aumento de datos**: Generar nuevas muestras a partir de las muestras con las que contamos.
- **Auto-encoders**: Agregar una etapa no supervisada.

Tipos de regularizaciones (algunos)

¿Cómo mejorar la capacidad de generalización del algoritmo?

- **Término de penalización (o regularizador)**: Incluir un término adicional en la función costo.
- **Inyección de ruido**: Incluir algún tipo de ruido en el proceso de aprendizaje.
- **Early stopping (parado temprano)**: Detener el proceso de aprendizaje tempranamente con algún criterio razonable.
- **Aumento de datos**: Generar nuevas muestras a partir de las muestras con las que contamos.
- **Auto-encoders**: Agregar una etapa no supervisada.

Para la regularización es clave el conjunto de validación!

Penalización

Término de penalización que perturba la optimización de la función costo:

$$J(\theta) = \frac{1}{B}\sum_{i=1}^{B} L_i(\theta) + \lambda R(\theta)$$

Weight decay or L2 regularization

$$J(\theta) = \frac{1}{B}\sum_{i=1}^{B} L_i(\theta) + \sum_{l=1}^{L} \frac{\lambda_l}{2} \cdot \|\mathbf{W}^{[l]}\|_F^2 \rightarrow \frac{1}{2}\frac{\partial \|\mathbf{W}^{[l]}\|_F^2}{\partial \mathbf{W}^{[l]}} = \mathbf{W}^{[l]}$$

Inyección de ruido

Se agrega ruido **ÚNICAMENTE** durante el entrenamiento.

Tipos de inyección de ruido

- En los parámetros (no tan habitual)
- En las entradas (relacionado con data-augmentation)
- **En las unidades ocultas**

Srivastava et al. 2014: "Dropout: A simple way to prevent neural networks from overfitting".

Inyección de ruido

Se agrega ruido **ÚNICAMENTE** durante el entrenamiento.

Tipos de inyección de ruido

- En los parámetros (no tan habitual)
- En las entradas (relacionado con data-augmentation)
- **En las unidades ocultas**

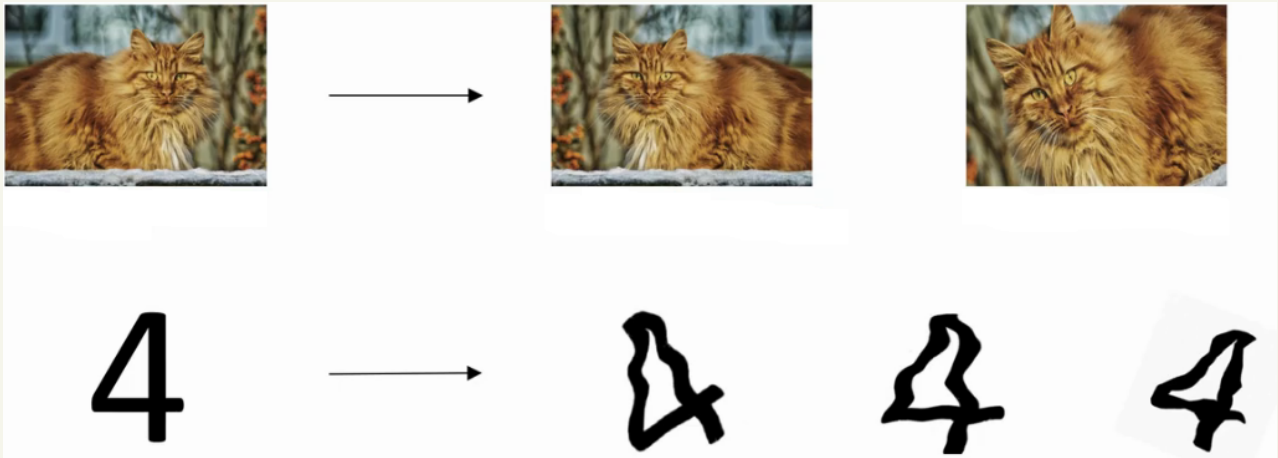
Dropout (inverted dropout technique)

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]} \left(\mathbf{h}^{[l-1]} \odot \frac{1}{p_{\text{keep}}^{[l-1]}} \boldsymbol{\eta}^{[l-1]} \right) + \mathbf{b}^{[l]}, \qquad \text{con } \boldsymbol{\eta}^{[l-1]} \sim \text{Ber}\left(p_{\text{keep}}^{[l-1]} \right)$$

Srivastava et al. 2014: "Dropout: A simple way to prevent neural networks from overfitting".

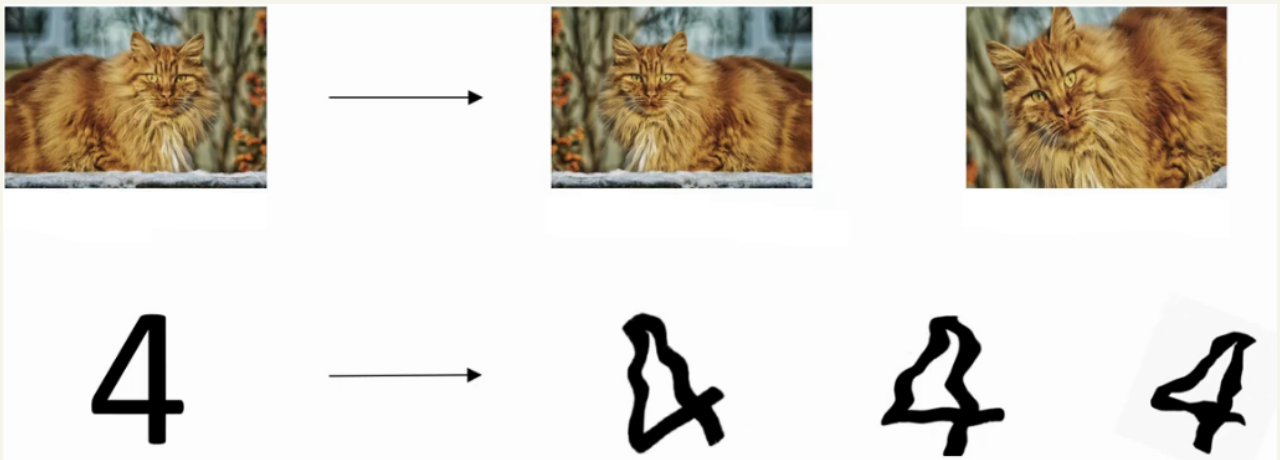
Data augmentation

Generar nuevas muestras inyectándole ruido a las que tenemos (noise inyection a la entrada). Normalmente se incrementa la cantidad de muestras (no se reemplazan unas por otras). Muy poderoso cuando se usa información adicional sobre como pueden variar las muestras.



Data augmentation

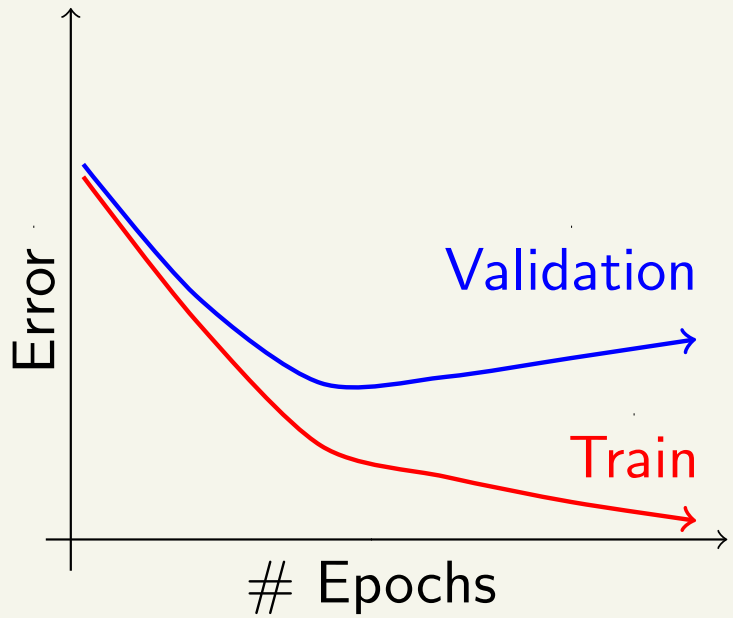
Generar nuevas muestras inyectándole ruido a las que tenemos (noise inyection a la entrada). Normalmente se incrementa la cantidad de muestras (no se reemplazan unas por otras). Muy poderoso cuando se usa información adicional sobre como pueden variar las muestras.



- Se incrementa la cantidad de muestras.
- Las nuevas muestras son más robustas a perturbaciones.
- Se pierde la independencia entre muestras.

Early stopping

Para de entrenar antes!

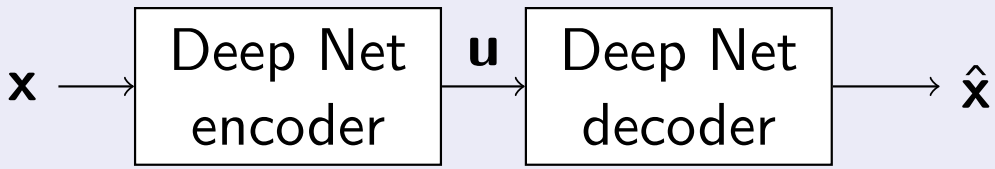


- PRO: Para en el punto óptimo para la validación.
- CONTRA: Impacta directamente en el bias.

Autoencoder

Método de aprendizaje no supervisado que busca generar reprsentaciones (típicamente de menor dimensión) al estilo PCA.

Diagrama en bloques



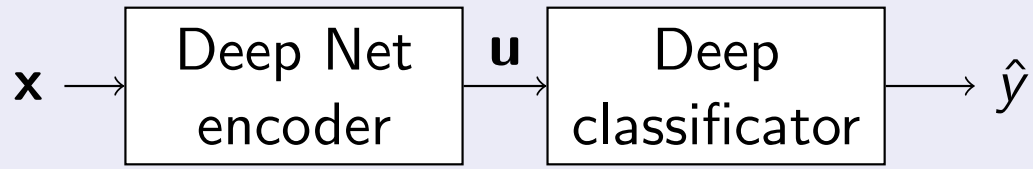
Función Costo

La función costo más habitual es el error cuadrático medio $\mathbb{E}\left[\|\mathbf{X} - \hat{\mathbf{X}}\|_2^2\right]$.

Autoencoder como regularización para la clasificación

Preprocesing: Opción 1

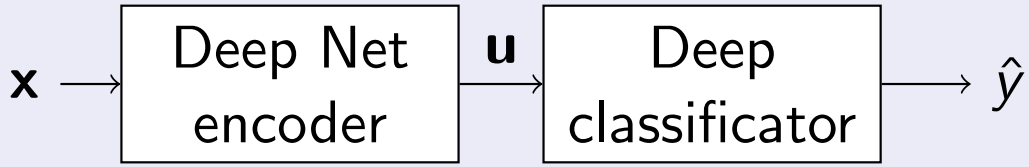
Entrenar el autoencoder y luego usar los **u** para entrenar el clasificador.



Autoencoder como regularización para la clasificación

Preprocesing: Opción 1

Entrenar el autoencoder y luego usar los **u** para entrenar el clasificador.



Preprocesing: Opción 2

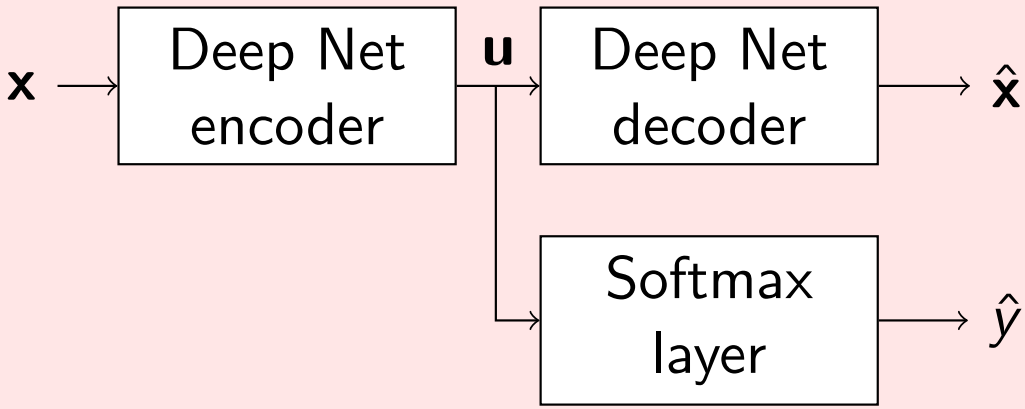
Entrenar el autoencoder y luego usar los $\hat{\mathbf{x}}$ para entrenar el clasificador.

Autoencoder como regularización para la clasificación

Regularizador: Opción 3

Entrenar simultáneamente ponderando las funciones costo

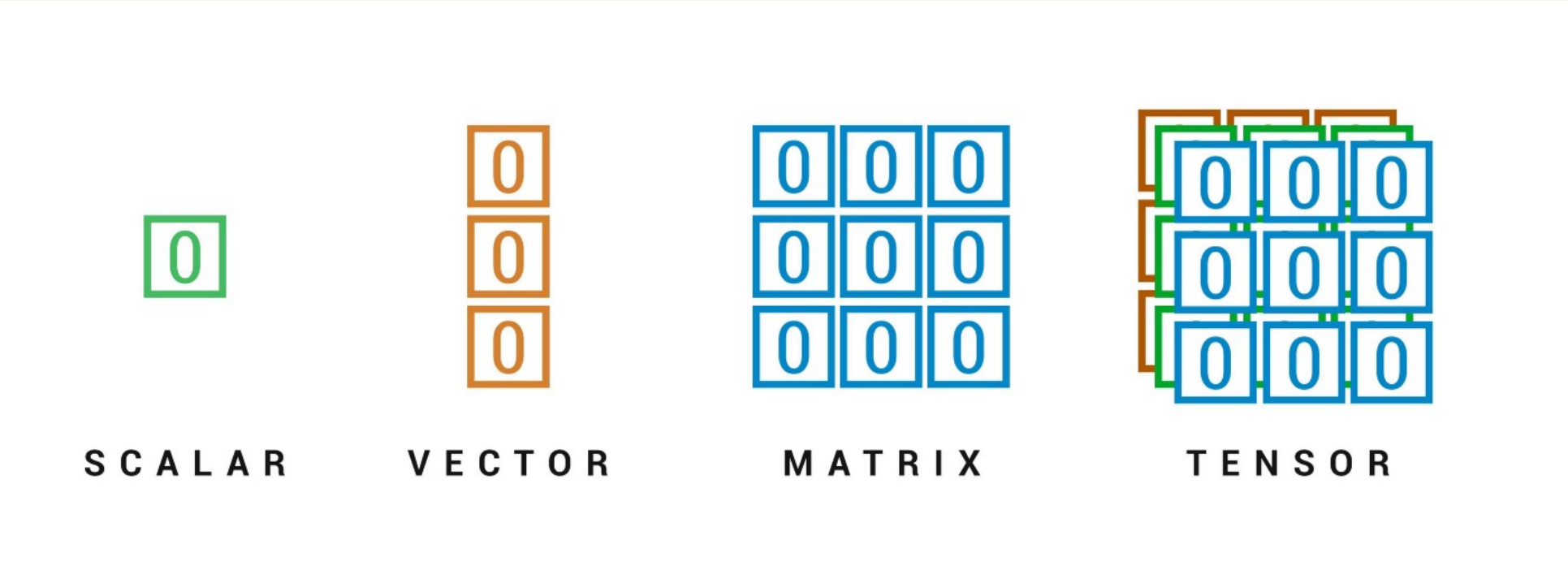
$J(\theta) = \beta J_{\text{clas}}(\theta) + (1 - \beta) J_{\text{ae}}(\theta)$



Agenda

1. Redes Neuronales y el Deep World
2. Entrenamiento de Redes Neuronales Profundas
3. Detalles de Deep Learning
4. Regularización
5. Redes Convolucionales
6. Redes Recurrentes

¿Que es un tensor?



ConvNets

Las redes neuronales convolucionales son redes que combinan los siguientes tipos de layers:

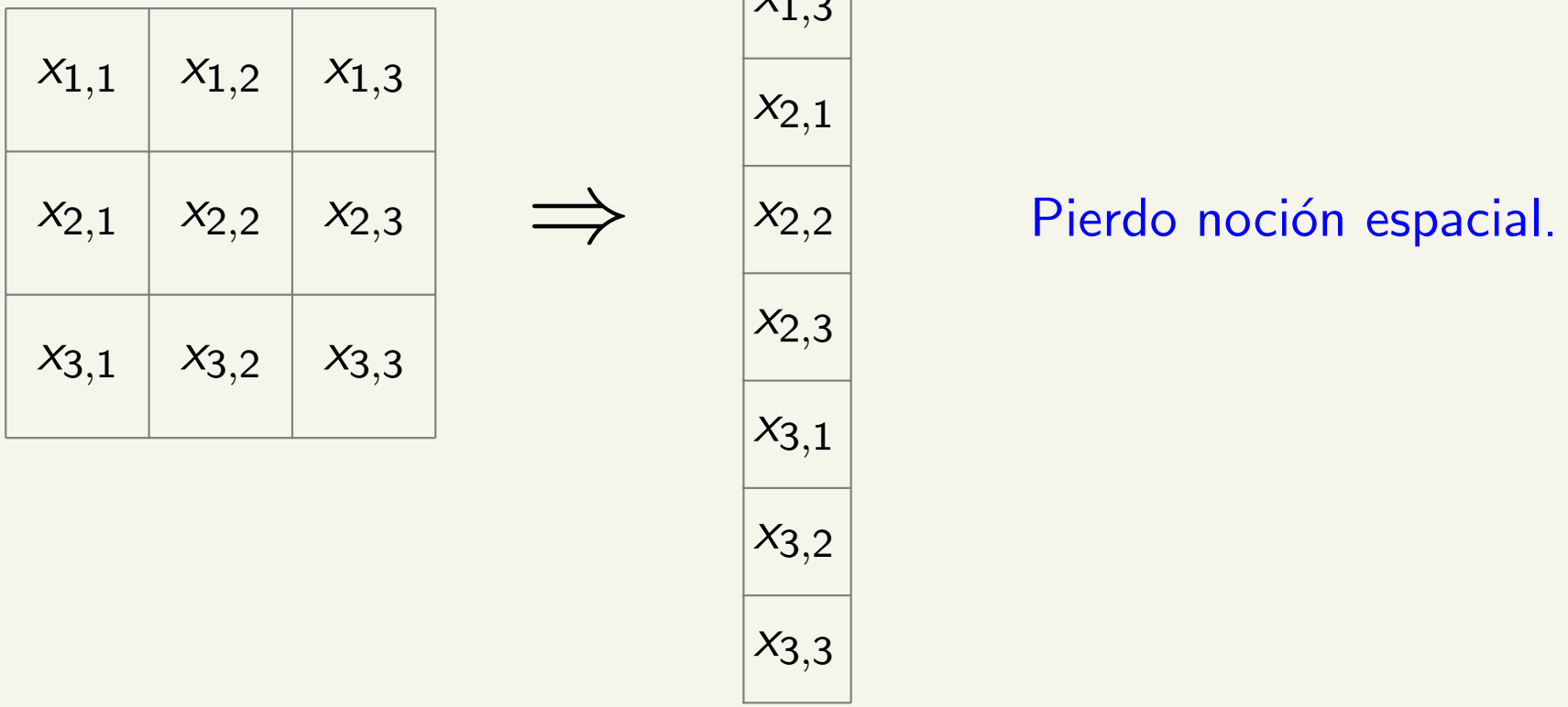
- Dense or Fully Conected.
- Flatten o Global Pooling.
- Convolutional.
- Pooling.
- Accesorios: ej. Dropout.

ConvNets

Las redes neuronales convolucionales son redes que combinan los siguientes tipos de layers:

- **Dense or Fully Conected.** ✓
- **Flatten o Global Pooling.** ✓
- Convolutional.
- Pooling.
- **Accesorios: ej. Dropout.** ✓

Flatten layer



Pierdo noción espacial.

Dense Layer

$$\mathbf{h}^{[l]} = \boldsymbol{\sigma}^{[l]}(\mathbf{W}^{[l]}\mathbf{h}^{[l-1]} + \mathbf{b}^{[l]})$$

$$\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & w_{1,5} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & w_{2,5} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & w_{3,5} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & w_{4,5} \end{bmatrix} \cdot \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

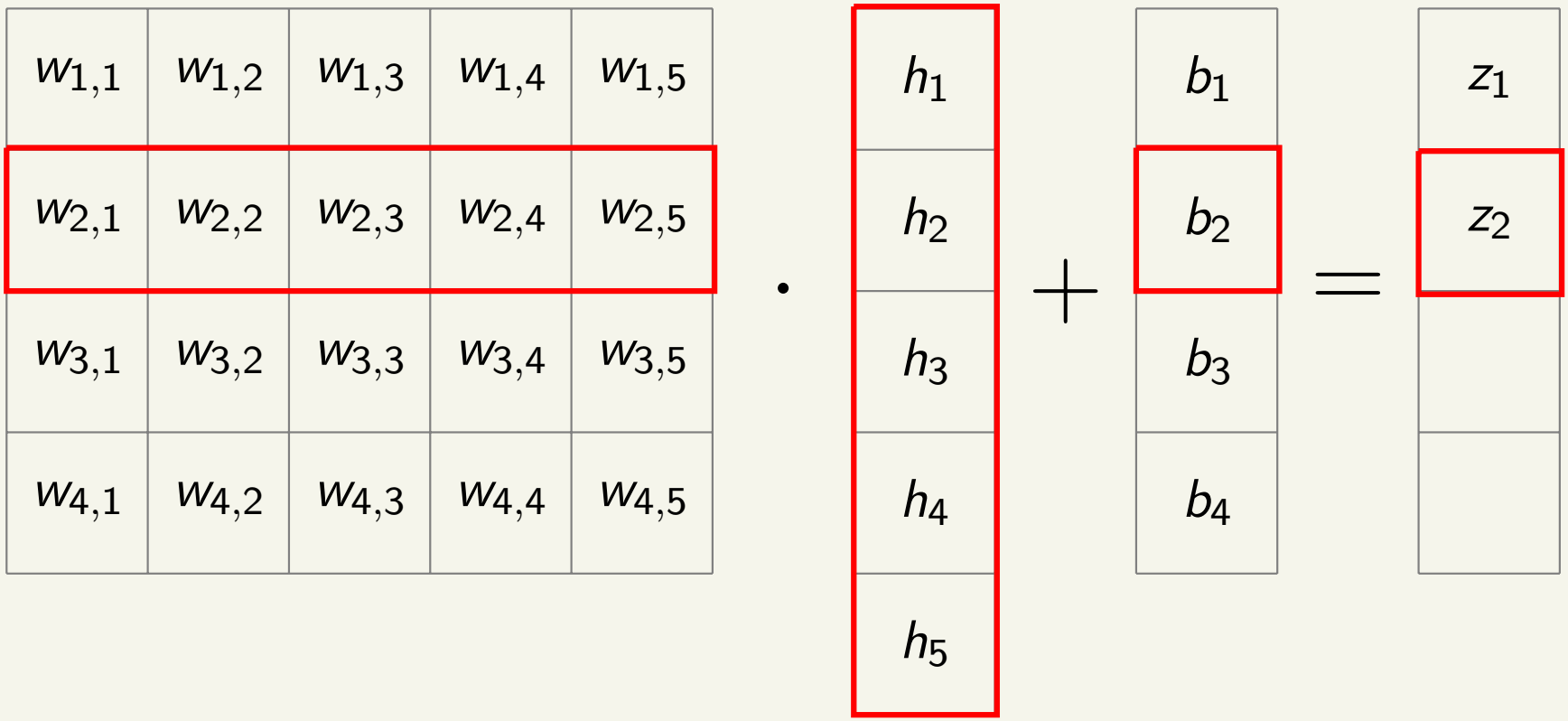
Dense Layer

$$\mathbf{h}^{[l]} = \boldsymbol{\sigma}^{[l]}(\mathbf{W}^{[l]}\mathbf{h}^{[l-1]} + \mathbf{b}^{[l]})$$

$$\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & w_{1,5} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & w_{2,5} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & w_{3,5} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & w_{4,5} \end{bmatrix} \cdot \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} z_1 \\ \\ \\ \end{bmatrix}$$

Dense Layer

$$\mathbf{h}^{[l]} = \boldsymbol{\sigma}^{[l]}(\mathbf{W}^{[l]}\mathbf{h}^{[l-1]} + \mathbf{b}^{[l]})$$



Dense Layer

$$\mathbf{h}^{[l]} = \boldsymbol{\sigma}^{[l]}(\mathbf{W}^{[l]}\mathbf{h}^{[l-1]} + \mathbf{b}^{[l]})$$

$$\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & w_{1,5} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & w_{2,5} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & w_{3,5} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & w_{4,5} \end{bmatrix} \cdot \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \end{bmatrix}$$

Dense Layer

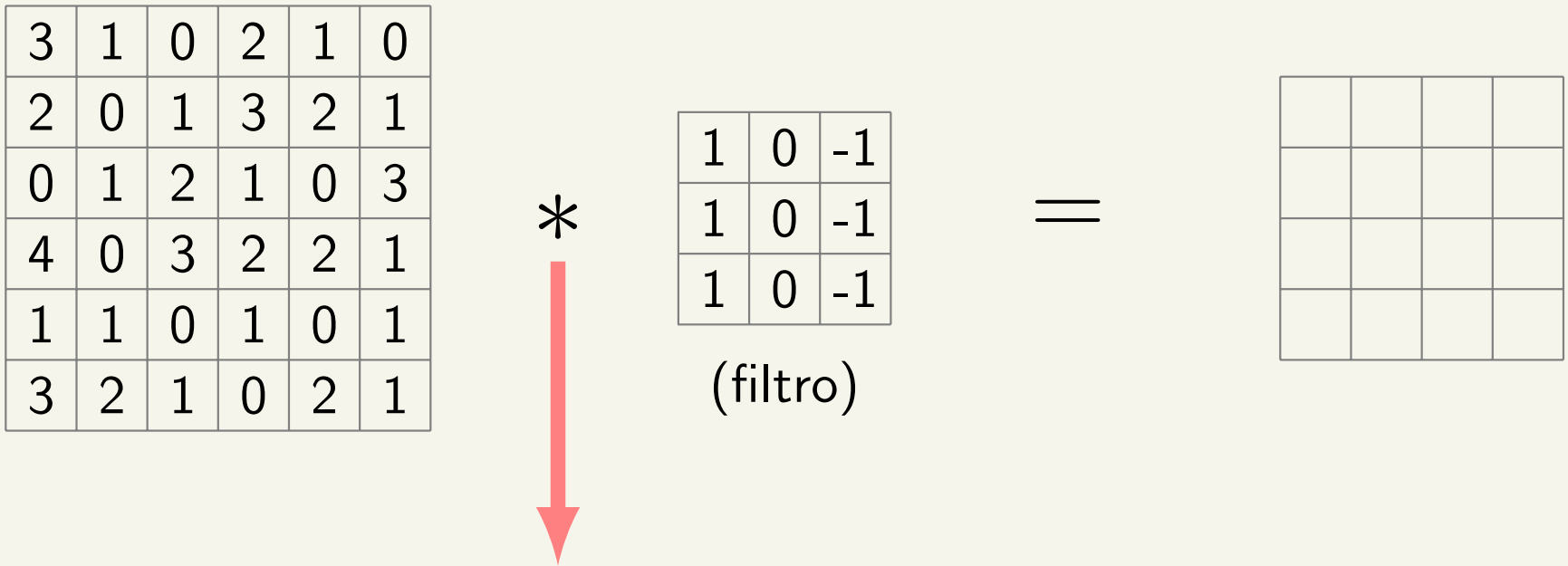
$$\mathbf{h}^{[l]} = \sigma^{[l]}(\mathbf{W}^{[l]}\mathbf{h}^{[l-1]} + \mathbf{b}^{[l]})$$

$$\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & w_{1,5} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & w_{2,5} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & w_{3,5} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & w_{4,5} \end{bmatrix} \cdot \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix}$$

Para generar cada neurona se necesita toda una fila de $\mathbf{W}^{[l]}$ (es decir una cantidad de parámetros igual a la dimensión de la entrada) y un solo elemento de $\mathbf{b}^{[l]}$.

Convolutional Layers

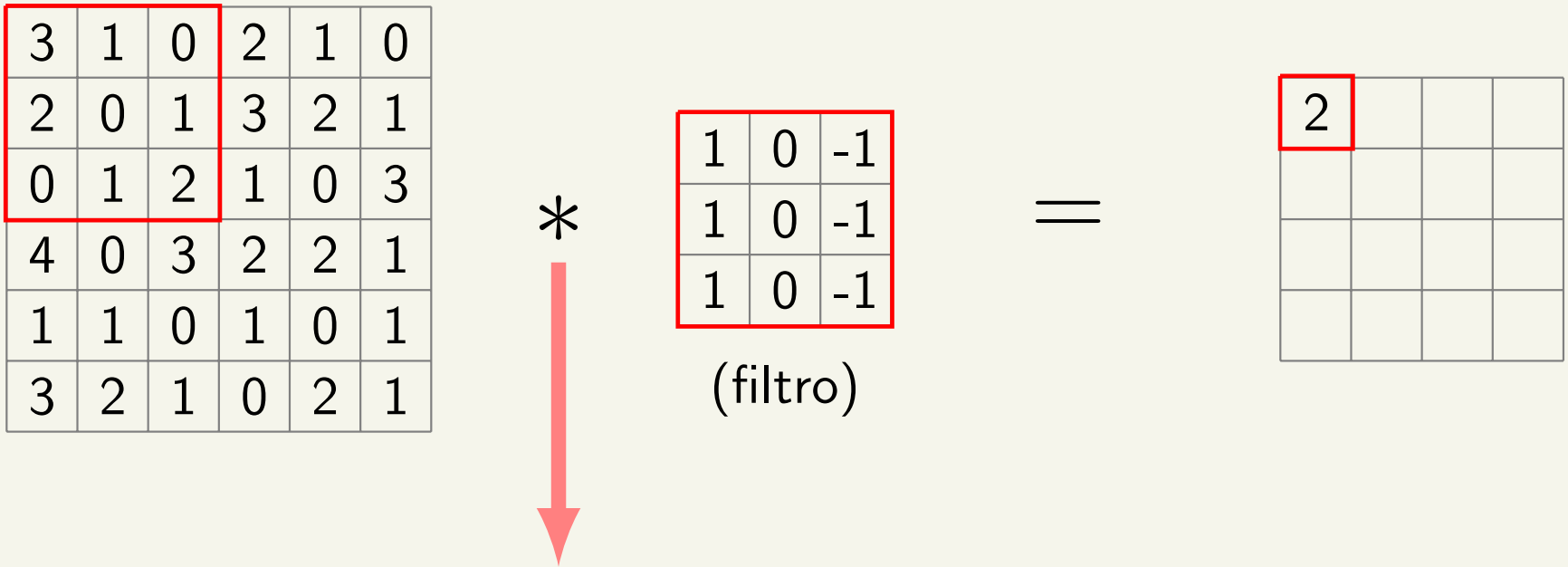
Un convolutional layer básicamente un reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.



La operación no espeja, se suele pensar que el filtro aprendido ya flipeo.

Convolutional Layers

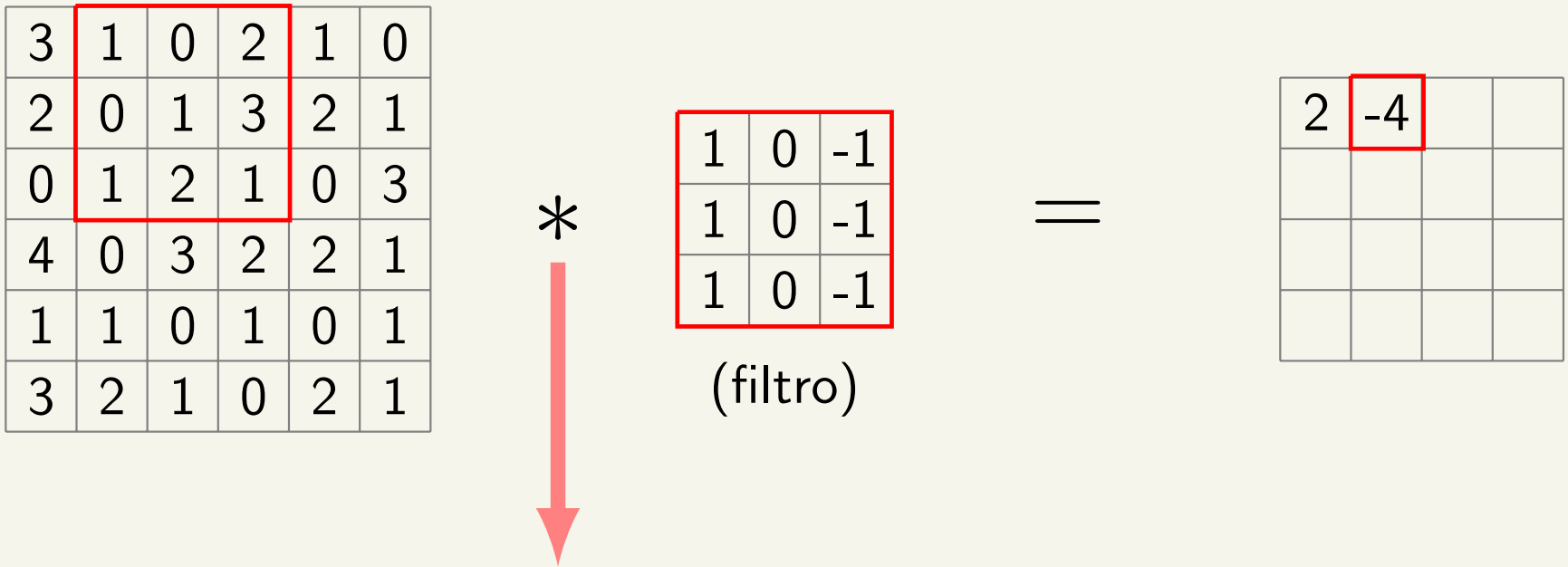
Un convolutional layer básicamente un reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.



La operación no espeja, se suele pensar que el filtro aprendido ya flipeo.

Convolutional Layers

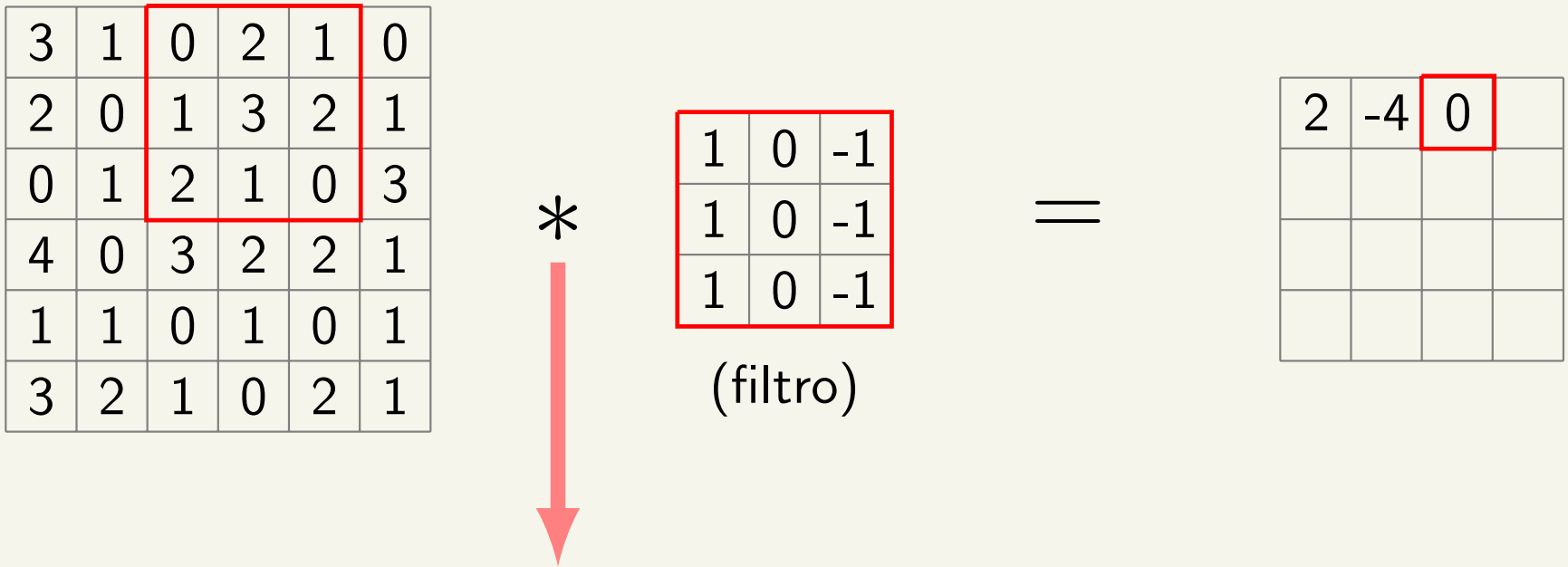
Un convolutional layer básicamente un reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.



La operación no espeja, se suele pensar que el filtro aprendido ya flipeo.

Convolutional Layers

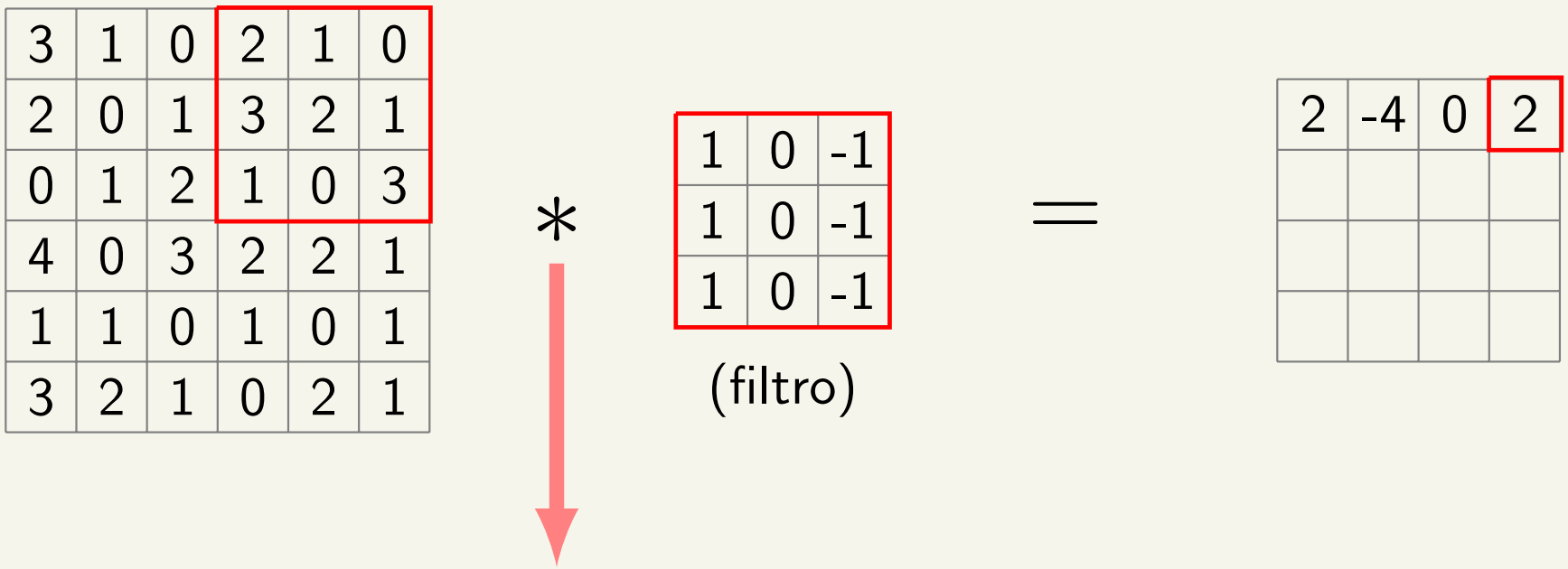
Un convolutional layer básicamente un reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.



La operación no espeja, se suele pensar que el filtro aprendido ya flipeo.

Convolutional Layers

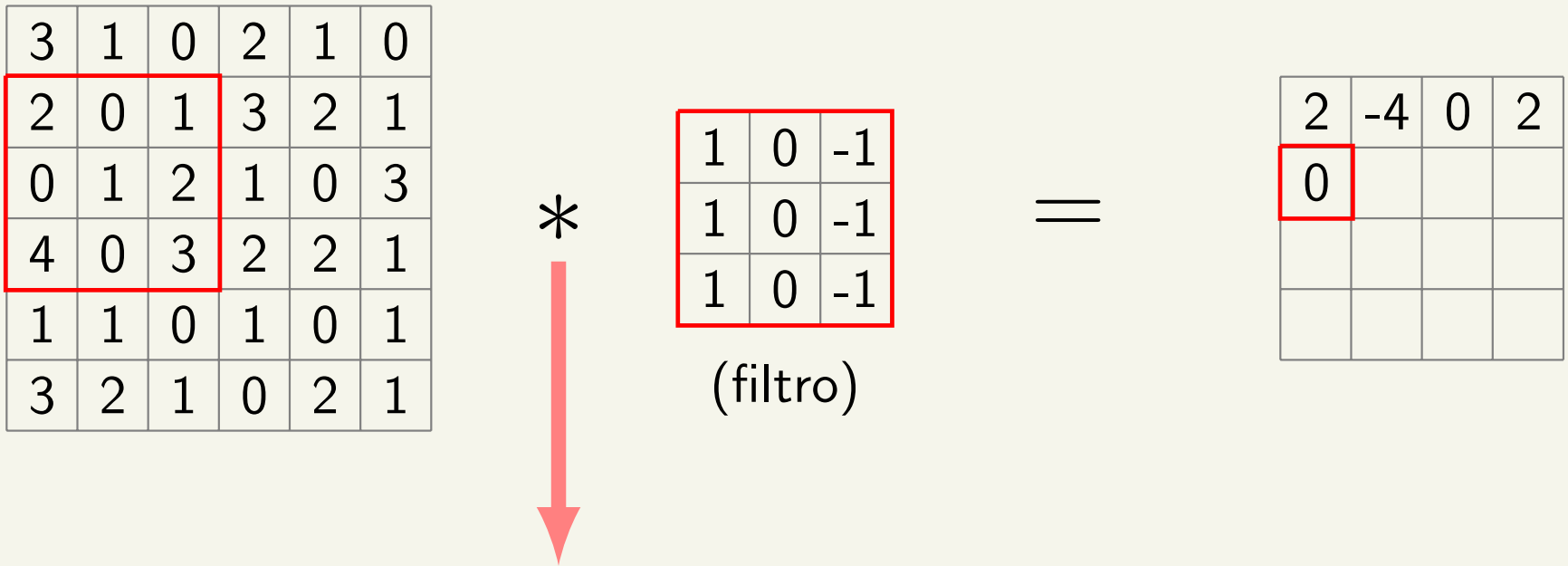
Un convolutional layer básicamente un reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.



La operación no espeja, se suele pensar que el filtro aprendido ya flipeo.

Convolutional Layers

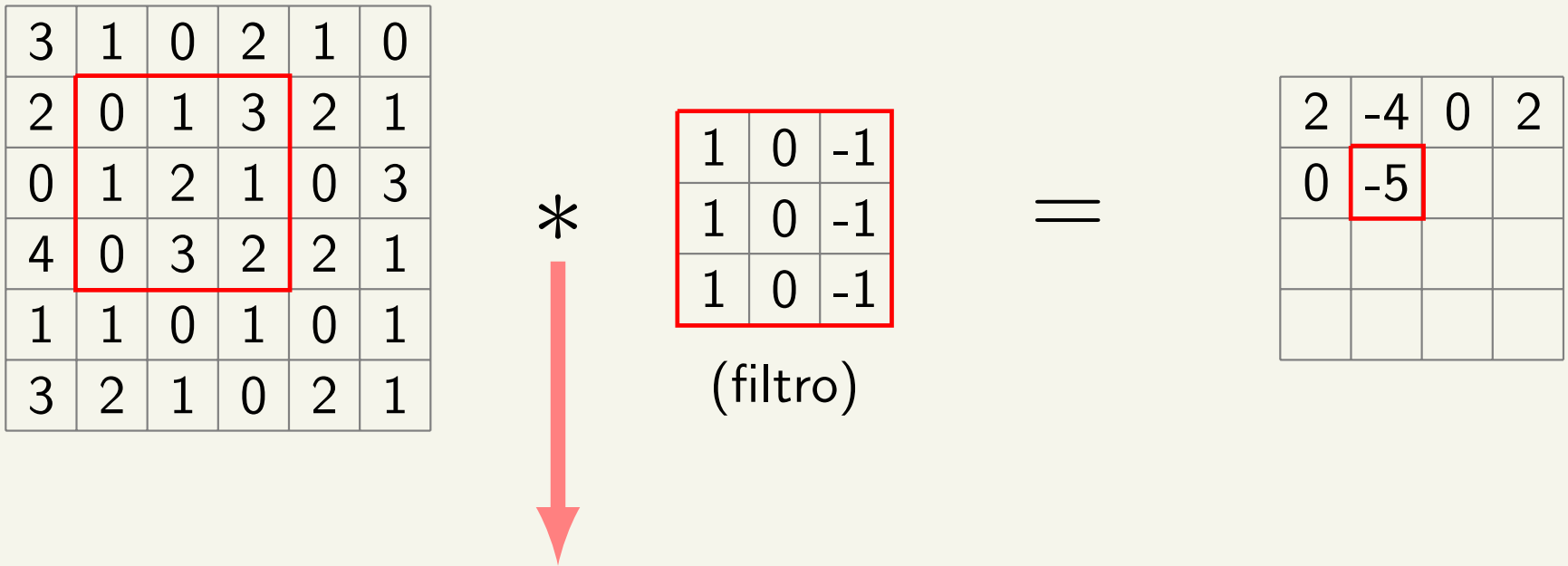
Un convolutional layer básicamente un reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.



La operación no espeja, se suele pensar que el filtro aprendido ya flipeo.

Convolutional Layers

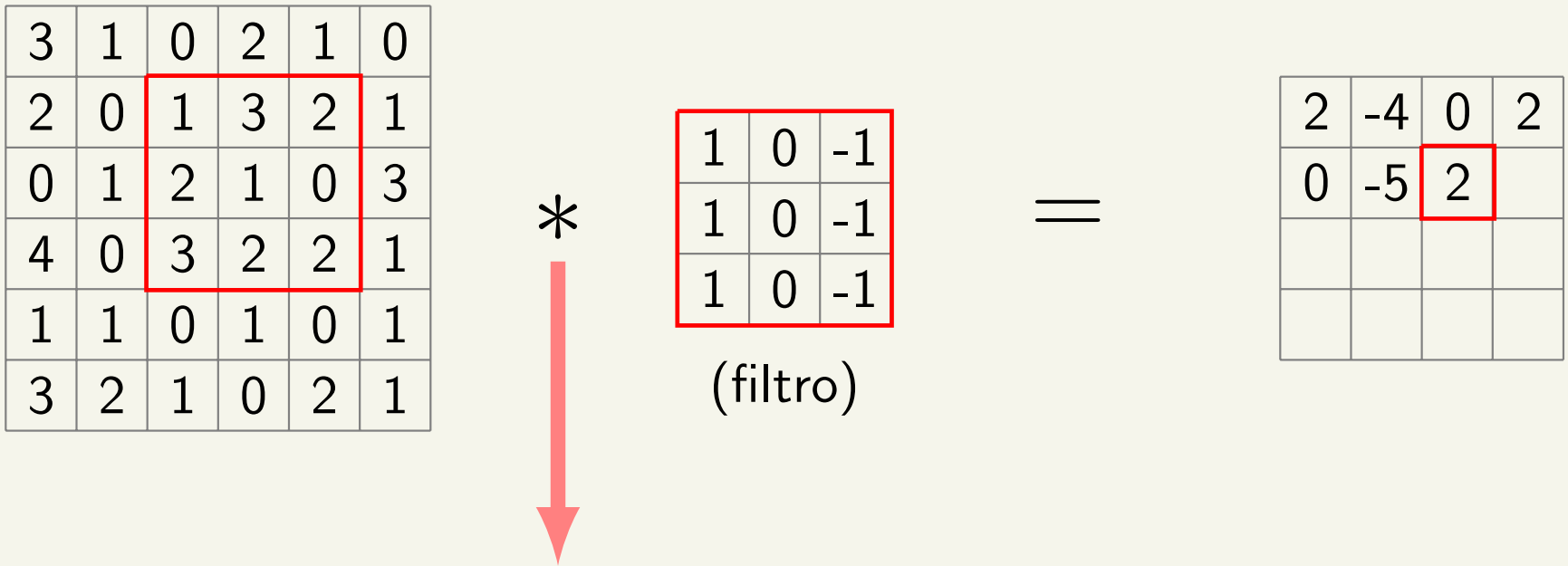
Un convolutional layer básicamente un reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.



La operación no espeja, se suele pensar que el filtro aprendido ya flipeo.

Convolutional Layers

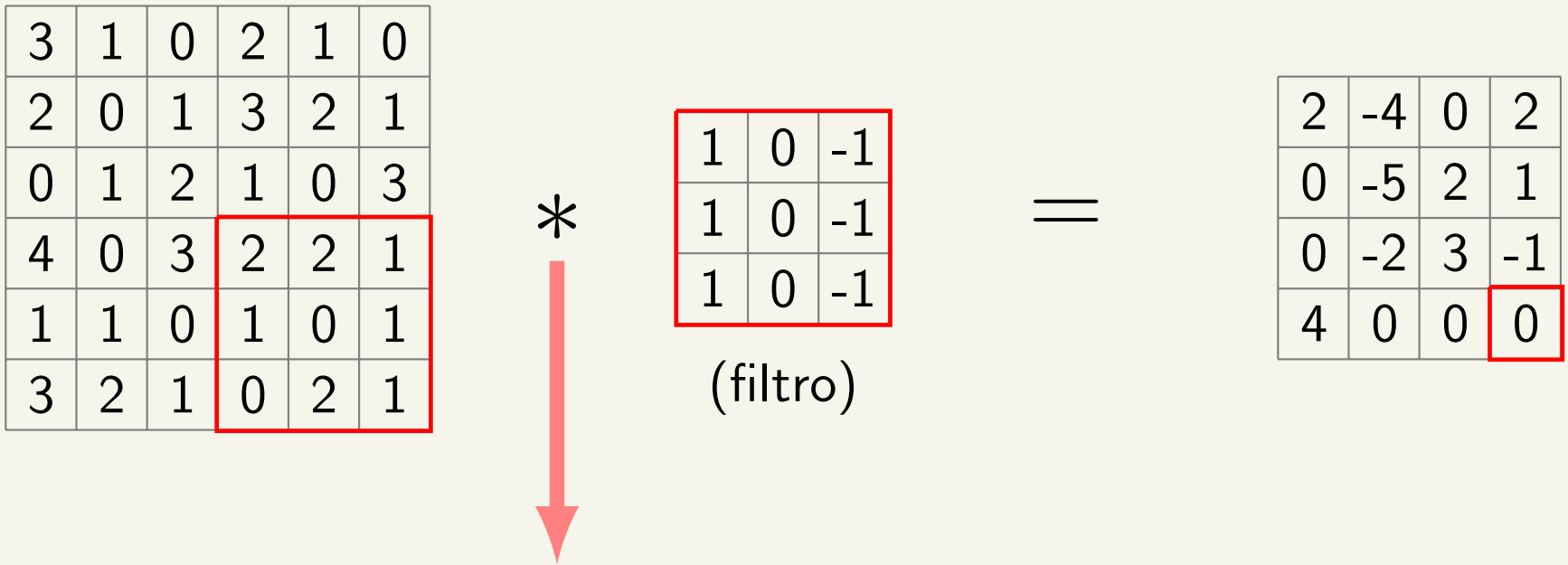
Un convolutional layer básicamente un reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.



La operación no espeja, se suele pensar que el filtro aprendido ya flipeo.

Convolutional Layers

Un convolutional layer básicamente un reemplaza la reducción dimensional del dense layer (producido al multiplicar una matriz por un vector) de manera que no se necesite que la entrada sea un vector y que utilice menos parámetros para dimensionar cada neurona.



La operación no espeja, se suele pensar que el filtro aprendido ya flipeo.

Convolutional layers: Características de cada neurona

$$\text{Neurona} \rightarrow \sigma(\mathbf{x} * \text{FILTRO} + b \cdot \mathbf{1})$$

- No pierde noción espacial (no necesita hacer flatten).
- Cada neurona generada es una matriz (en lugar de un escalar).
- Necesita menos parámetros (en el ejemplo para generar una neurona a partir de una entradas de $6 \times 6 = 36$ bastaron con $3 \times 3 = 9$ parámetros).
- A pesar que la salida del filtrado es una matriz, se suele usar un solo escalar por neurona para el bias (al igual que el dense).

Convolutional layers: Características de cada neurona

$$\text{Neurona} \rightarrow \sigma(\mathbf{x} * \text{FILTRO} + b \cdot \mathbf{1})$$

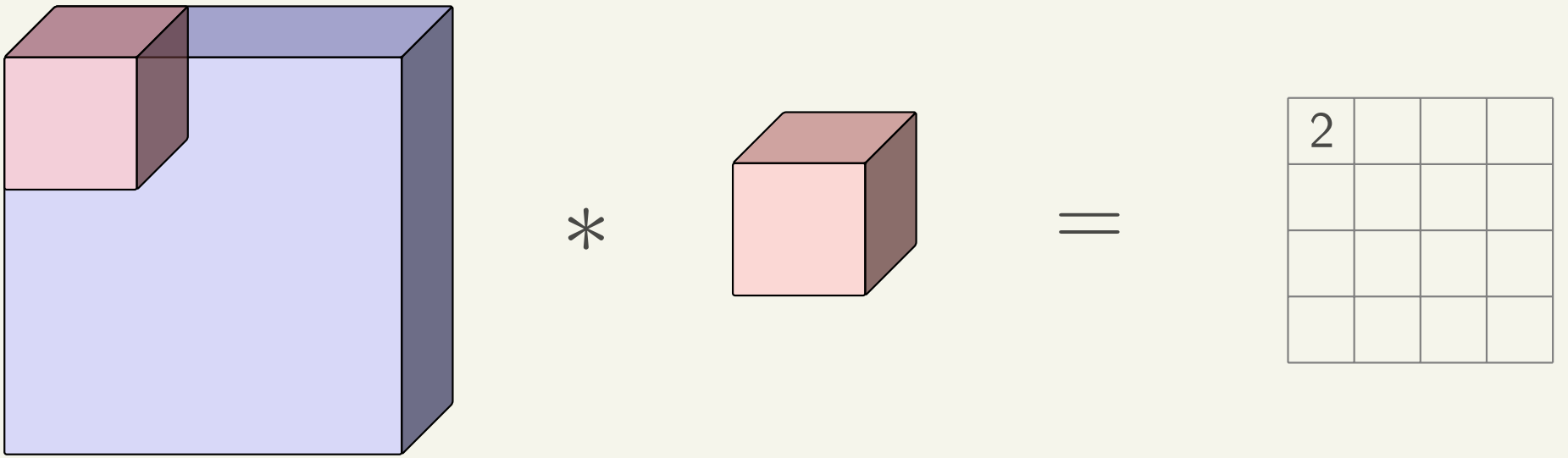
- No pierde noción espacial (no necesita hacer flatten).
- Cada neurona generada es una matriz (en lugar de un escalar).
- Necesita menos parámetros (en el ejemplo para generar una neurona a partir de una entradas de $6 \times 6 = 36$ bastaron con $3 \times 3 = 9$ parámetros).
- A pesar que la salida del filtrado es una matriz, se suele usar un solo escalar por neurona para el bias (al igual que el dense).

Sobre los filtros

Por un contexto histórico, relacionado con el mundo de *computer vision*, los filtros suelen tener el mismo alto que ancho y ser una dimensión impar (1×1, 3×3, 5×5, etc.).

$$\dim(\text{Neurona}) = \dim(\mathbf{x}) - \dim(\text{Filtro}) + 1$$

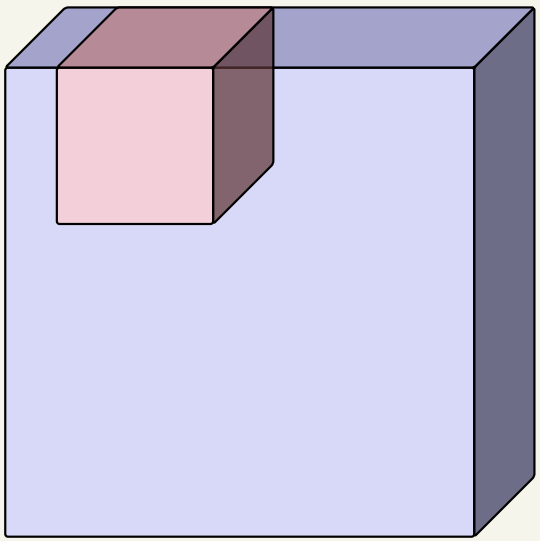
¿Que hacer cuando la entrada es un tensor?



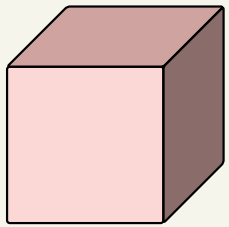
Recordar

No solamente las neuronas serán bidimensionales, sino que tengo muchas de ellas. Es decir, las unidades ocultas viven en $(n, 4, 4, d)$, donde d es la cantidad de filtros utilizados (cada uno con su bias).

¿Que hacer cuando la entrada es un tensor?



$*$



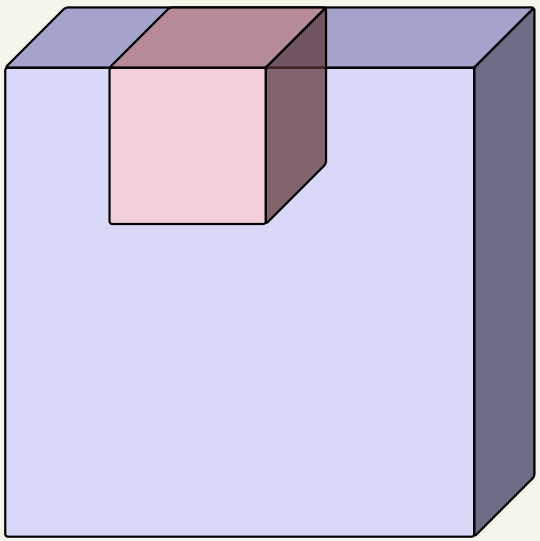
$=$

2	7		

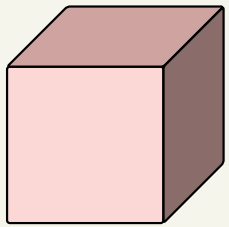
Recordar

No solamente las neuronas serán bidimensionales, sino que tengo muchas de ellas. Es decir, las unidades ocultas viven en $(n, 4, 4, d)$, donde d es la cantidad de filtros utilizados (cada uno con su bias).

¿Que hacer cuando la entrada es un tensor?



$*$



$=$

2	7	-1	

Recordar

No solamente las neuronas serán bidimensionales, sino que tengo muchas de ellas. Es decir, las unidades ocultas viven en $(n, 4, 4, d)$, donde d es la cantidad de filtros utilizados (cada uno con su bias).

Hiperparámetros de los layers convolucionales: Padding

Por un lado la dimensión a lo largo de la red va a ir disminuyendo (en redes grandes esto es problemático), y por el otro los números en el borde de la entrada se usan mucho menos que el resto (posible pérdida de información). Es por esto que surge el padding:

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1

$$\text{Padding} = 0$$

Hiperparámetros de los layers convolucionales: Padding

Por un lado la dimensión a lo largo de la red va a ir disminuyendo (en redes grandes esto es problemático), y por el otro los números en el borde de la entrada se usan mucho menos que el resto (posible pérdida de información). Es por esto que surge el padding:

0	0	0	0	0	0	0	0
0	3	1	0	2	1	0	0
0	2	0	1	3	2	1	0
0	0	1	2	1	0	3	0
0	4	0	3	2	2	1	0
0	1	1	0	1	0	1	0
0	3	2	1	0	2	1	0
0	0	0	0	0	0	0	0

$$\text{Padding} = 1$$

Hiperparámetros de los layers convolucionales: Padding

Por un lado la dimensión a lo largo de la red va a ir disminuyendo (en redes grandes esto es problemático), y por el otro los números en el borde de la entrada se usan mucho menos que el resto (posible pérdida de información). Es por esto que surge el padding:

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	3	1	0	2	1	0	0	0
0	0	2	0	1	3	2	1	0	0
0	0	0	1	2	1	0	3	0	0
0	0	4	0	3	2	2	1	0	0
0	0	1	1	0	1	0	1	0	0
0	0	3	2	1	0	2	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$\text{Padding} = 2$$

$$\dim(\text{Neurona}) = \dim(\mathbf{x}) - \dim(\text{Filtro}) + 1 + 2 \cdot \text{Padding}$$

Tipos de padding habituales

- *Valid*: Padding $= 0$.
- *Same*: Padding $= \frac{\text{dim(Filtro)}-1}{2}$.
- *Full*: Padding $= \text{dim(Filtro)} - 1$.

Tipos de padding habituales

- *Valid*: Padding $= 0$.
- *Same*: Padding $= \frac{\dim(\text{Filtro})-1}{2}$.
- *Full*: Padding $= \dim(\text{Filtro}) - 1$.

Padding Same

El objetivo de padding same es que la neurona tenga la misma dimensión que la entrada $\dim(\text{Neurona}) = \dim(\mathbf{x})$.

Tipos de padding habituales

- *Valid*: Padding $= 0$.
- *Same*: Padding $= \frac{\dim(\text{Filtro})-1}{2}$.
- *Full*: Padding $= \dim(\text{Filtro}) - 1$.

Padding Same

El objetivo de padding same es que la neurona tenga la misma dimensión que la entrada $\dim(\text{Neurona}) = \dim(\mathbf{x})$.

Padding Full

Padding full es el mayor padding que tiene sentido (sin agregar ceros triviales). No suele alcanzar buenols resultados, por lo que rara vez se utiliza.

Stride

Ya sea por eficiencia computacional o porque deseamos reducir las dimensiones, podemos decidir mover la ventana más de un elemento a la vez, omitiendo las ubicaciones intermedias. Esto es especialmente útil si el kernel es grande, ya que captura una gran área de la imagen subyacente.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1

$*$

1	0	-1
1	0	-1
1	0	-1

(filtro)

$=$

Stride

Ya sea por eficiencia computacional o porque deseamos reducir las dimensiones, podemos decidir mover la ventana más de un elemento a la vez, omitiendo las ubicaciones intermedias. Esto es especialmente útil si el kernel es grande, ya que captura una gran área de la imagen subyacente.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1

$*$

1	0	-1
1	0	-1
1	0	-1

(filtro)

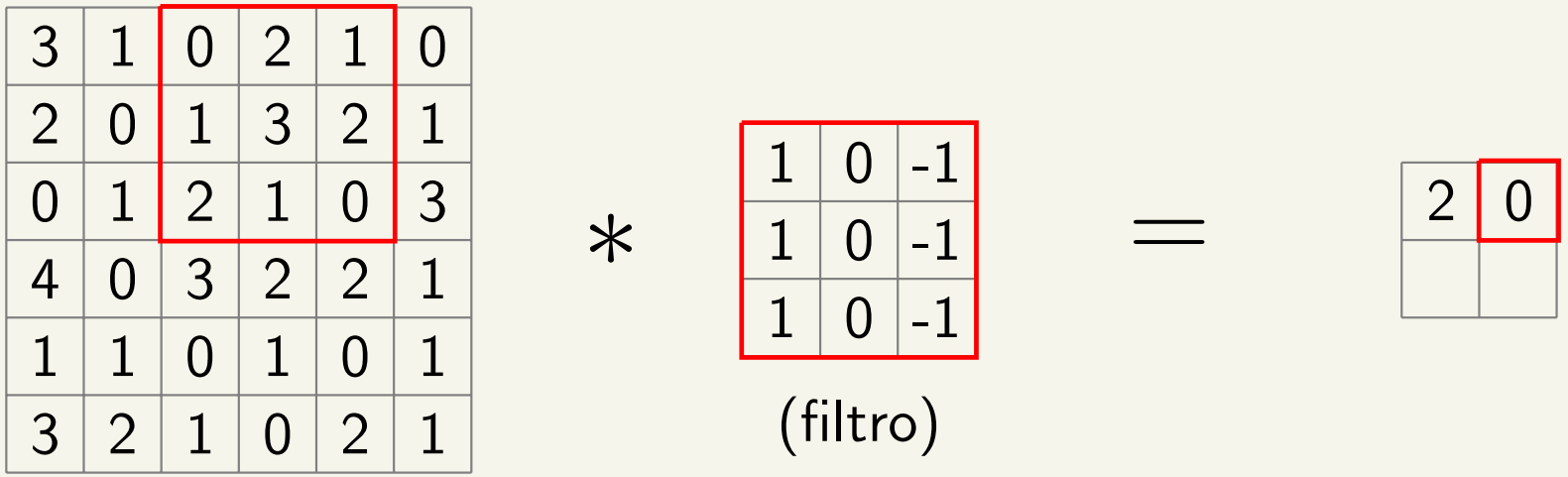
$=$

2	

Stride $= 2$

Stride

Ya sea por eficiencia computacional o porque deseamos reducir las dimensiones, podemos decidir mover la ventana más de un elemento a la vez, omitiendo las ubicaciones intermedias. Esto es especialmente útil si el kernel es grande, ya que captura una gran área de la imagen subyacente.



Stride = 2

Stride

Ya sea por eficiencia computacional o porque deseamos reducir las dimensiones, podemos decidir mover la ventana más de un elemento a la vez, omitiendo las ubicaciones intermedias. Esto es especialmente útil si el kernel es grande, ya que captura una gran área de la imagen subyacente.

3	1	0	2	1	0
2	0	1	3	2	1
0	1	2	1	0	3
4	0	3	2	2	1
1	1	0	1	0	1
3	2	1	0	2	1

$*$

1	0	-1
1	0	-1
1	0	-1

(filtro)

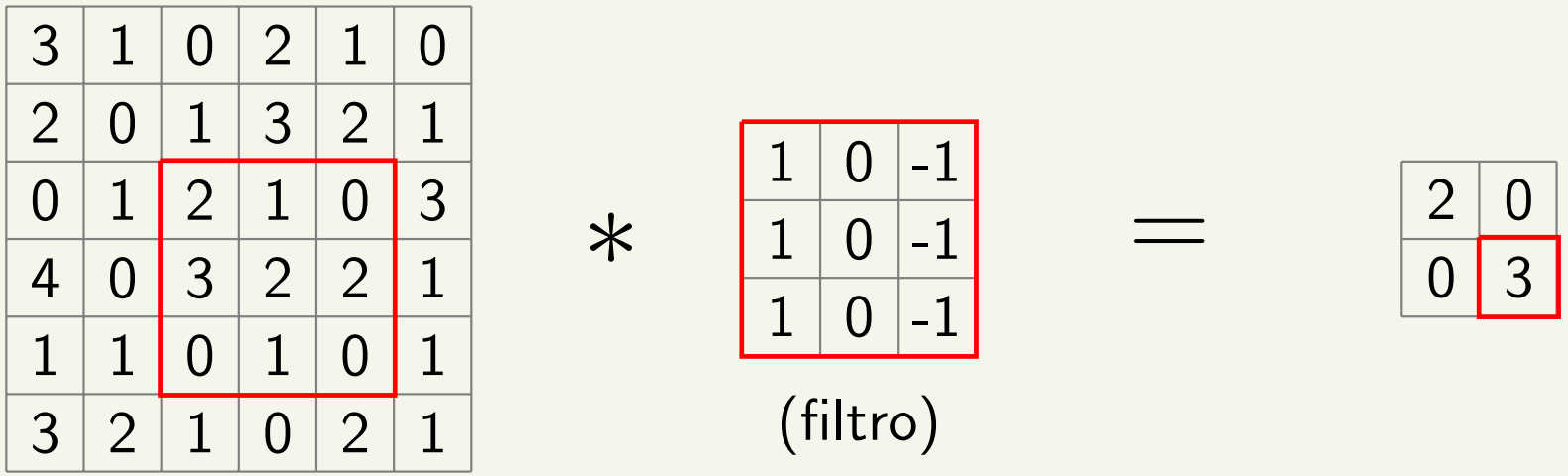
$=$

2	0
0	

Stride $= 2$

Stride

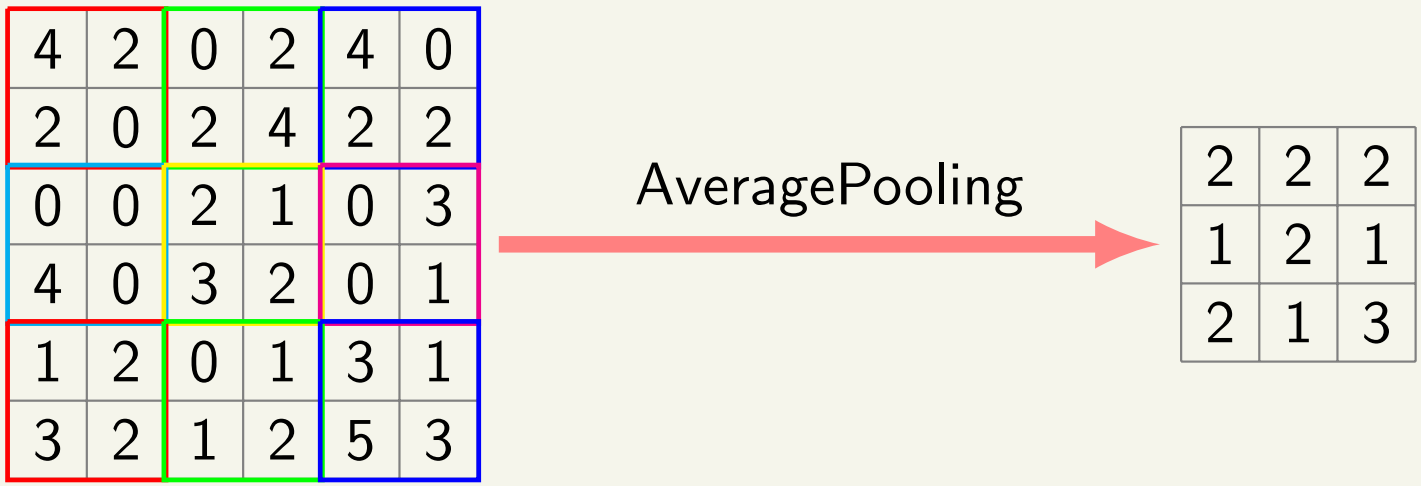
Ya sea por eficiencia computacional o porque deseamos reducir las dimensiones, podemos decidir mover la ventana más de un elemento a la vez, omitiendo las ubicaciones intermedias. Esto es especialmente útil si el kernel es grande, ya que captura una gran área de la imagen subyacente.



Stride = 2

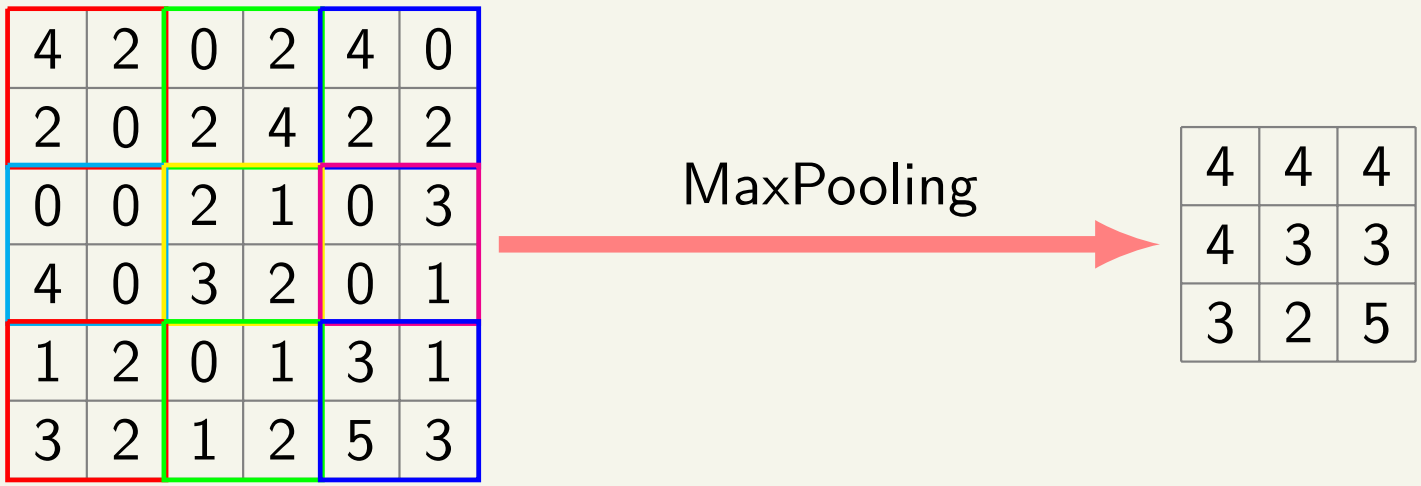
Pooling

El uso de Pooling puede entenderse como usar la información conocida de que la clasificación de imágenes no depende de rotaciones o traslaciones pequeñas.



Pooling

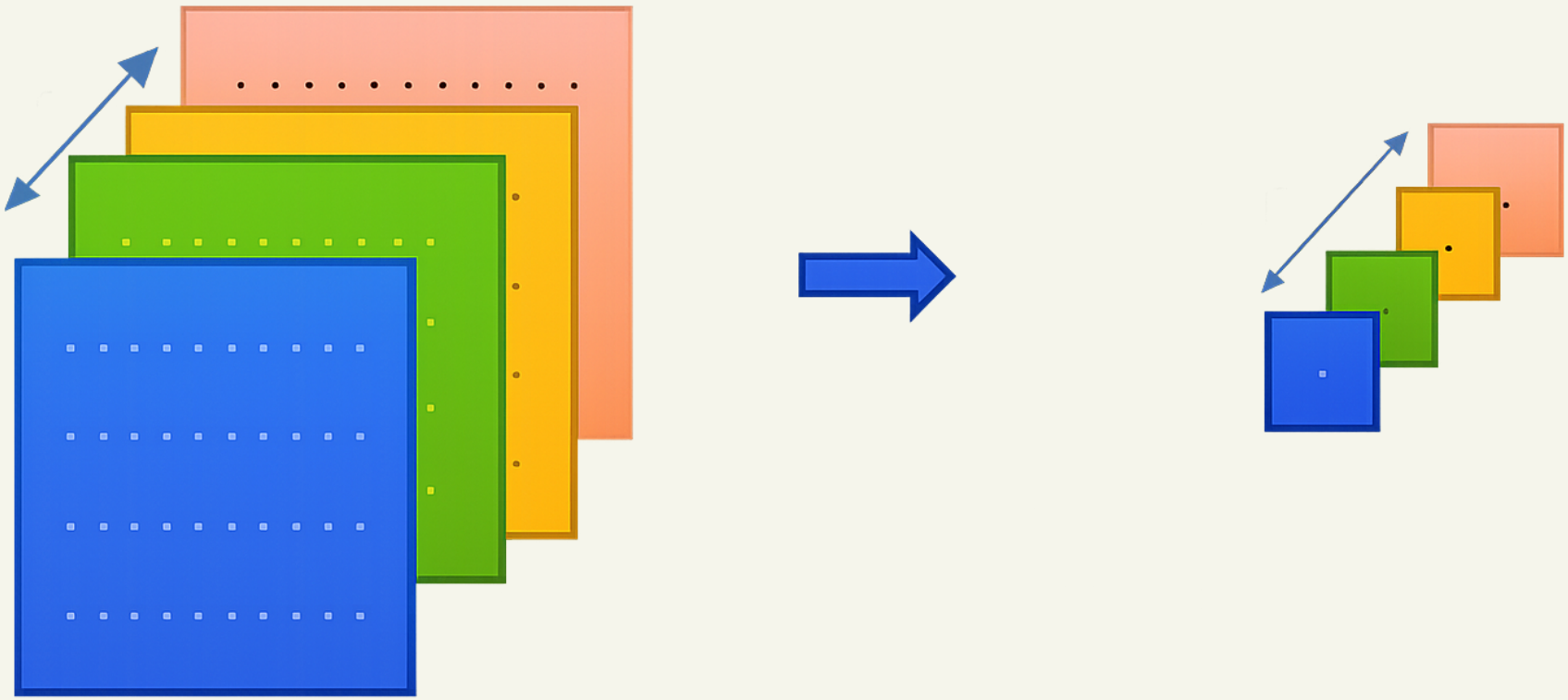
El uso de Pooling puede entenderse como usar la información conocida de que la clasificación de imágenes no depende de rotaciones o traslaciones pequeñas.



Global Pooling

Global Pooling

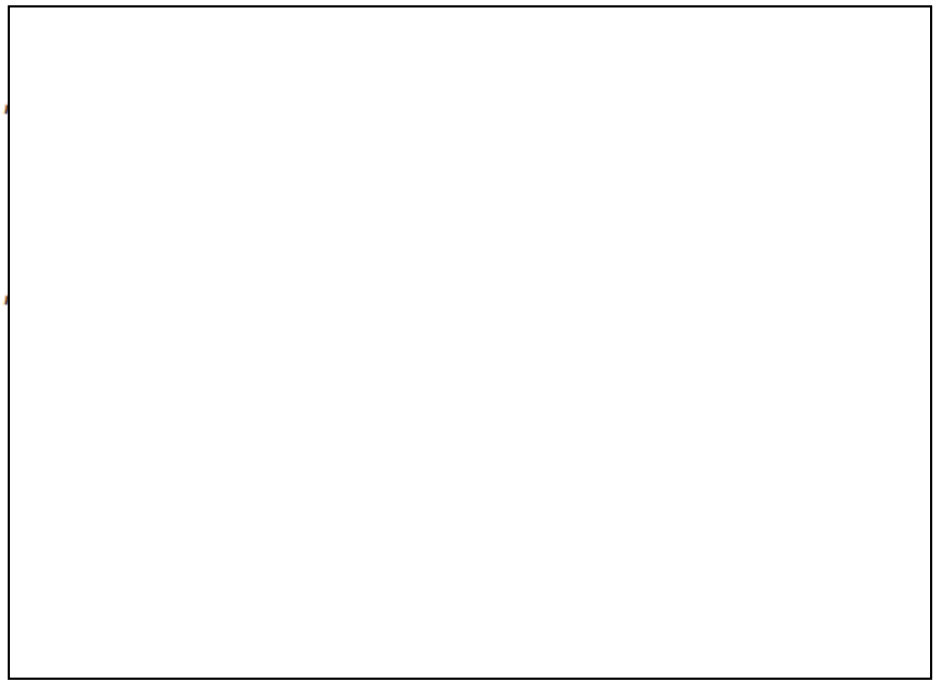
Para una tarea de clasificación/regresión, finalizar el bloque convolucional con un global pooling puede ser superador a hacerlo con un *flatten*. El pooling nos permite que las muestras tengan diferentes largos.



¿Como chequear si entendí CNN? Ej. Lenet

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Conv2D(filters=6, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(tf.keras.layers.AveragePooling2D())
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=3, activation='relu'))
model.add(tf.keras.layers.AveragePooling2D())
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units=120, activation='relu'))
model.add(tf.keras.layers.Dense(units=84, activation='relu'))
model.add(tf.keras.layers.Dense(units=10, activation = 'softmax'))
model.summary()
```

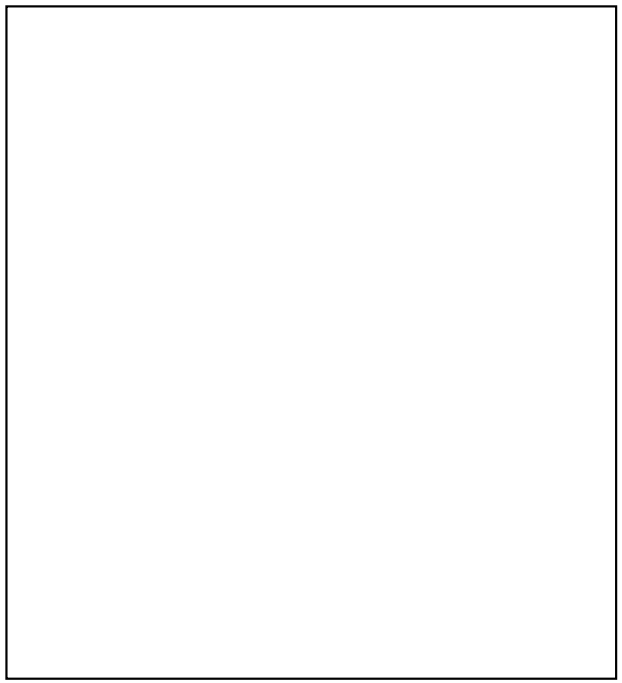
```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_17 (Conv2D)           (None,

 average_pooling2d_14 (Avera  (None,
 gePooling2D)

 conv2d_18 (Conv2D)           (None,

 average_pooling2d_15 (Avera  (None,
 gePooling2D)

 flatten_7 (Flatten)          (None,

 dense_21 (Dense)             (None,

 dense_22 (Dense)             (None,

 dense_23 (Dense)             (None,
```



¿Como chequear si entendí CNN? Ej. Lenet

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Conv2D(filters=6, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(tf.keras.layers.AveragePooling2D())
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=3, activation='relu'))
model.add(tf.keras.layers.AveragePooling2D())
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units=120, activation='relu'))
model.add(tf.keras.layers.Dense(units=84, activation='relu'))
model.add(tf.keras.layers.Dense(units=10, activation = 'softmax'))
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_17 (Conv2D)          (None, 26, 26, 6)

 average_pooling2d_14 (Avera  (None, 13, 13, 6)
 gePooling2D)

 conv2d_18 (Conv2D)          (None, 11, 11, 16)

 average_pooling2d_15 (Avera  (None, 5, 5, 16)
 gePooling2D)

 flatten_7 (Flatten)         (None, 400)

 dense_21 (Dense)            (None, 120)

 dense_22 (Dense)            (None, 84)

 dense_23 (Dense)            (None, 10)
```



¿Como chequear si entendí CNN? Ej. Lenet

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.Conv2D(filters=6, kernel_size=3, activation='relu', input_shape=(28,28,1)))
model.add(tf.keras.layers.AveragePooling2D())
model.add(tf.keras.layers.Conv2D(filters=16, kernel_size=3, activation='relu'))
model.add(tf.keras.layers.AveragePooling2D())
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(units=120, activation='relu'))
model.add(tf.keras.layers.Dense(units=84, activation='relu'))
model.add(tf.keras.layers.Dense(units=10, activation = 'softmax'))
model.summary()
```

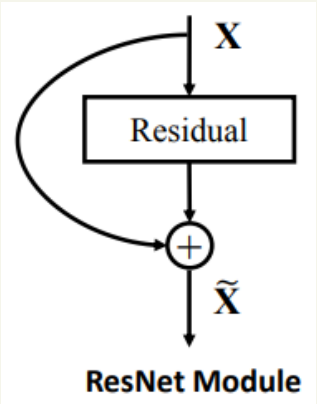
```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_17 (Conv2D)          (None, 26, 26, 6)         60

 average_pooling2d_14 (Avera  (None, 13, 13, 6)        0
 gePooling2D)

 conv2d_18 (Conv2D)          (None, 11, 11, 16)        880

 average_pooling2d_15 (Avera  (None, 5, 5, 16)         0
 gePooling2D)

 flatten_7 (Flatten)         (None, 400)               0

 dense_21 (Dense)            (None, 120)               48120

 dense_22 (Dense)            (None, 84)                10164

 dense_23 (Dense)            (None, 10)                850
```

Redes Residuales

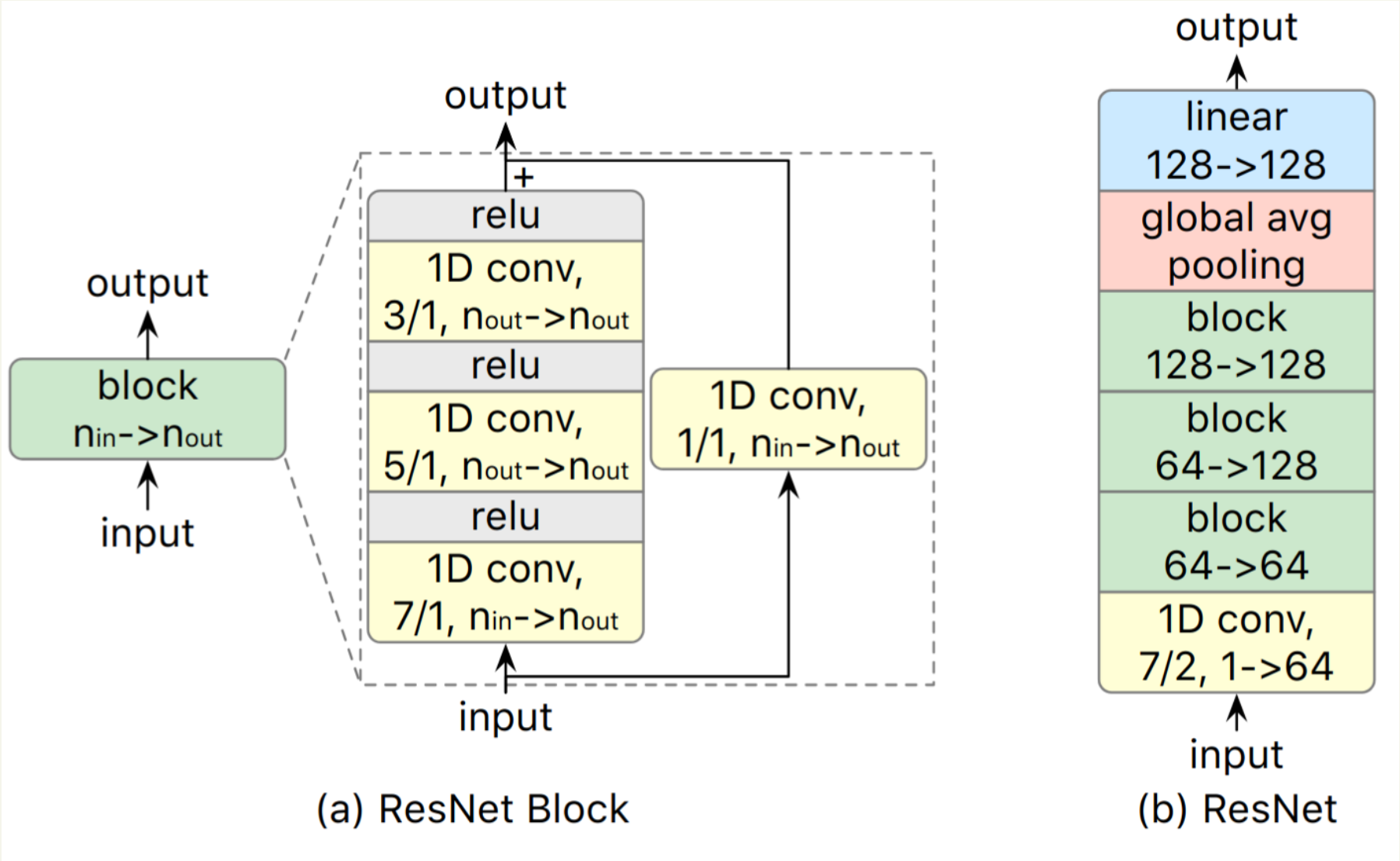
Residual Connections

Suele ser más fácil aprender una corrección sobre la entrada que la totalidad de ella. Es decir, sea $F(x)$ la representación de un bloque, en muchos casos conviene definir a la salida como $G(x) = x + F(x)$.

- Tolerancia al *vanishing* del gradiente, ya que $G'(x) = 1 + F'(x)$.
- Presencia de la información original (por ejemplo si $F(x) \approx 0$, $G(x) \approx x$).



Redes Residuales



Yeh et al 2023: "Toward a Foundation Model for Time Series Data".

Agenda

1. Redes Neuronales y el Deep World
2. Entrenamiento de Redes Neuronales Profundas
3. Detalles de Deep Learning
4. Regularización
5. Redes Convolucionales
6. Redes Recurrentes

Redes Recurrentes

Motivación

Las redes convolucionales poseen una memoria relacionada con el tamaño del kernel. Para lograr capturar la dependencia de largo alcance es necesario algún tipo de memoria dinámica.

Redes Recurrentes

Las redes recurrentes constituyen una arquitectura capaz de capturar la dependencia local de forma comparable a las redes convolucionales, pero con acceso a memoria lejana. A diferencia de las convolucionales, fueron diseñadas desde un principio para aplicarse en series de tiempo.

Función de Activación

En estas arquitecturas no se descartan activaciones populares como la ReLU, pero el estándar es la tanh ya que se necesita un rango de valores acotados para evitar explosiones y tener la capacidad de representar valores positivos y negativos.

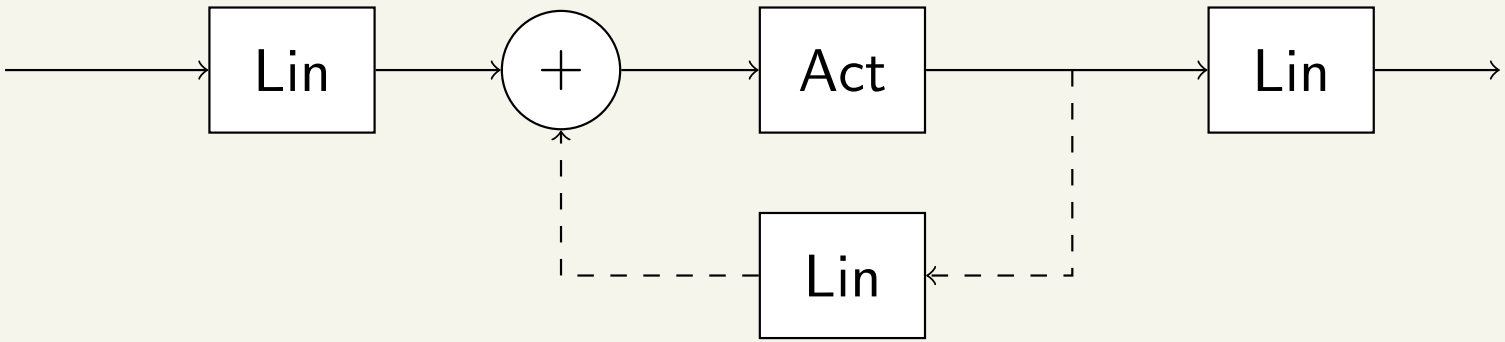
Redes Recurrentes

Modelo

Las redes neuronales recurrentes (RNN) extienden a las redes tradicionales al introducir un *estado oculto que se actualiza en el tiempo*, lo que les permite procesar secuencias y capturar dependencias temporales. Su funcionamiento básico se resume en:

- $h_t = \sigma(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$
- $y_t = W_{hy}h_t + b_y$.

donde x_t es la entrada en el instante t, h_t es el estado oculto (la *memoria* de la red), y_t es la salida y $\sigma(\cdot)$ una función de activación.



Redes Recurrentes

Incialización problemática

La matriz W_{hh} debe ser inicializada con cuidado para que el gradiente no termine desvaneciéndose o explotando, ya que este se computa pasando una vez por cada instante de tiempo por esa matriz. La estabilidad puede lograrse utilizando inicializaciones ortogonales.

Inicialización Ortogonal

Sea W una matriz cuadrada tal que $W^T \cdot W = I$. Luego, la transformación $W \cdot h$ preserva la norma:

$$\|W \cdot h\|^2 = h^T \cdot W^T \cdot W \cdot h = \|h\|^2$$

Desvaneciéndose y explción del gradiente

Sin embargo, esta técnica solo afecta al paso inicial, y con las iteraciones puede ser un problema la estabilidad del gradiente en estas arquitecturas.

Tratamiento temporal

Feature Representation

- **Flatten**. Se utiliza todo, requiere muchos parámetros y la necesidad de fijar largo de las secuencias.
- **Global Pooling**. Quedarse con el valor medio. Es la solución estándar en redes convolucionales.
- **Estado final**. Quedarse con el último instante de tiempo. Es la solución estándar en redes recurrentes.

¿Entendimos las RNN?

```python
model = models.Sequential([
    layers.Input(shape=(12,1)),
    layers.SimpleRNN(16, activation="relu"),
    layers.Dense(1)])

model.compile(optimizer="adam", loss="mse")
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)		
dense_1 (Dense)		

¿Entendimos las RNN?

```
model = models.Sequential([
    layers.Input(shape=(12,1)),
    layers.SimpleRNN(16, activation="relu"),
    layers.Dense(1)])

model.compile(optimizer="adam", loss="mse")
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 16)	
dense_1 (Dense)	(None, 1)	

¿Entendimos las RNN?

```
model = models.Sequential([
    layers.Input(shape=(12,1)),
    layers.SimpleRNN(16, activation="relu"),
    layers.Dense(1)])

model.compile(optimizer="adam", loss="mse")
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 16)	288
dense_1 (Dense)	(None, 1)	17

Redes Bidireccionales

Redes Bidireccionales

En aplicaciones no *forecasting*, puede que tengamos acceso al futuro de la señal (no solo al pasado). En esos casos se recomienda usar redes bidireccionales. Basta con plantear una red recurrente causal y otra anticausal para finalmente concatenar las salidas.

Ejemplo

Un ejemplo clásico es ECG, donde se desea clasificar cada latido. Dicha clasificación no es online, por lo que uno tiene acceso a toda la señal (o a un tramo largo de ella).

¿Entendimos las Bidireccionales?

```
model = models.Sequential([
    layers.Input(shape=(X.shape[1], X.shape[2])),
    layers.Bidirectional(layers.SimpleRNN(16, activation='relu')),
    layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='BinaryCrossentropy', metrics=['accuracy'])
model.summary()
```

Model: "sequential"

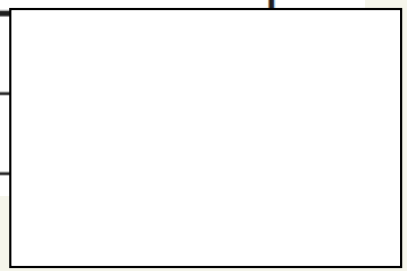
Layer (type)	Output Shape	Param #
bidirectional (Bidirectional)		
dense (Dense)		

¿Entendimos las Bidireccionales?

```
model = models.Sequential([
    layers.Input(shape=(X.shape[1], X.shape[2])),
    layers.Bidirectional(layers.SimpleRNN(16, activation='relu')),
    layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='BinaryCrossentropy', metrics=['accuracy'])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
bidirectional (Bidirectional)	(None, 32)	
dense (Dense)	(None, 1)	



¿Entendimos las Bidireccionales?

```
model = models.Sequential([
    layers.Input(shape=(X.shape[1], X.shape[2])),
    layers.Bidirectional(layers.SimpleRNN(16, activation='relu')),
    layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam', loss='BinaryCrossentropy', metrics=['accuracy'])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
bidirectional (Bidirectional)	(None, 32)	576
dense (Dense)	(None, 1)	33

Large Time Series Models

Debilidades de la RNN

- Cuando la secuencia es larga, los gradientes pueden desaparecer o explotar.
- Las RNN tienden a olvidar información importante de pasos previos.

Características de los LTSM

- Las LTSM proponen incorporar variables asociadas a celdas de memoria, las cuales serán controladas mediante compuertas.
- Entendemos por compuerta un limitador de caudal. Suelen usarse funciones sigmoide, donde 0 representa bloqueo de información y 1 transferencia máxima.

Large Time Series Models

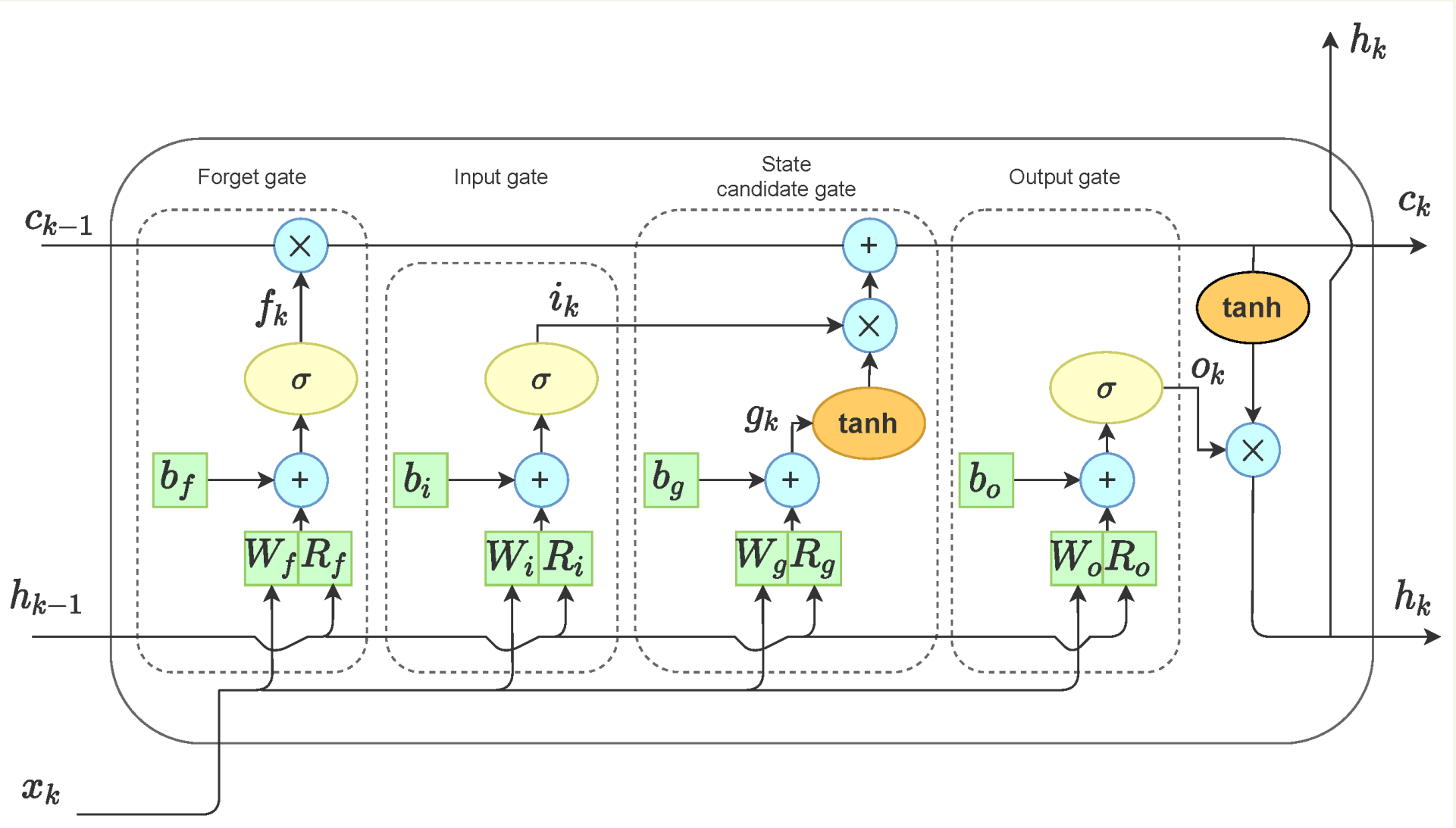
Modelo

Las redes LTSM incorporan 3 compuertas f_t (forget u olvido), o_t (output o estado actual) y i_t (input o novedad), y una celda de memoria.

- $f_t = \sigma(W_f x_t + R_f h_{t-1} + b_f)$ (forget gate)
- $i_t = \sigma(W_i x_t + R_i h_{t-1} + b_i)$ (input gate)
- $g_t = \tanh(W_g x_t + R_g h_{t-1} + b_g)$ (candidate cell state)
- $c_t = f_t \odot c_{t-1} + i_t \odot g_t$ (new cell state)
- $o_t = \sigma(W_o x_t + R_o h_{t-1} + b_o)$ (output gate)
- $h_t = o_t \odot \tanh(c_t)$ (hidden state)

donde $\sigma(\cdot)$ es la función sigmoide y $\odot$ es el producto elemento a elemento (Hadamard). La tanh puede ser reemplazada por otra función activación pero es el estándar.

Large Time Series Models



¿Entendimos las LSTM?

```
model = keras.Sequential([
    layers.Input(shape=(100, 1)),
    layers.LSTM(32),              # LSTM procesa la secuencia
    layers.Dense(3, activation='softmax')  # Clasificación
])
```

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)		
dense_1 (Dense)		

¿Entendimos las LSTM?

```
model = keras.Sequential([
    layers.Input(shape=(100, 1)),
    layers.LSTM(32),              # LSTM procesa la secuencia
    layers.Dense(3, activation='softmax')  # Clasificación
])
```

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 32)	
dense_1 (Dense)	(None, 3)	

¿Entendimos las LSTM?

```
model = keras.Sequential([
    layers.Input(shape=(100, 1)),
    layers.LSTM(32),              # LSTM procesa la secuencia
    layers.Dense(3, activation='softmax')  # Clasificación
])
```

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 32)	4,352
dense_1 (Dense)	(None, 3)	99

Gated Recurrent Unit

Debilidades de la LTSM

- Necesita demasiados parámetros, lo que vuelve la red lenta de entrenar y puede ocacionar problemas de overfitting.
- LSTM separa "qué olvidar" y "qué actualizar" en dos compuertas distintas; en ciertas tareas, esa separación no aporta beneficio real y puede incluso entorpecer la propagación de gradiente.

Características de las GRU

- No posee celda de memoria por separado, sino que está implícita en el esatdo oculto.
- En lugar de 3 compuertas utiliza 2: Update gate (decide cuánto del estado anterior se mantiene vs. reemplaza con nueva información) y Reset gate (controla cuánto del estado anterior influye en el candidato).

Gated Recurrent Unit

Modelo

El modelo GRU posee:

- $z_t = \sigma(W_z x_t + R_z h_{t-1} + b_z)$ (update gate)
- $r_t = \sigma(W_r x_t + R_r h_{t-1} + b_r)$ (reset gate)
- $g_t = \tanh(W_g x_t + R_g(r_t \odot h_{t-1}) + b_g)$ (candidate hidden state)
- $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot g_t$ (new hidden state)

donde $\sigma(\cdot)$ es la función sigmoide y $\odot$ es el producto elemento a elemento (Hadamard). La tanh puede ser reemplazada por otra función activación pero es el estándar.

Gated Recurrent Unit

Modelo

El modelo GRU posee:

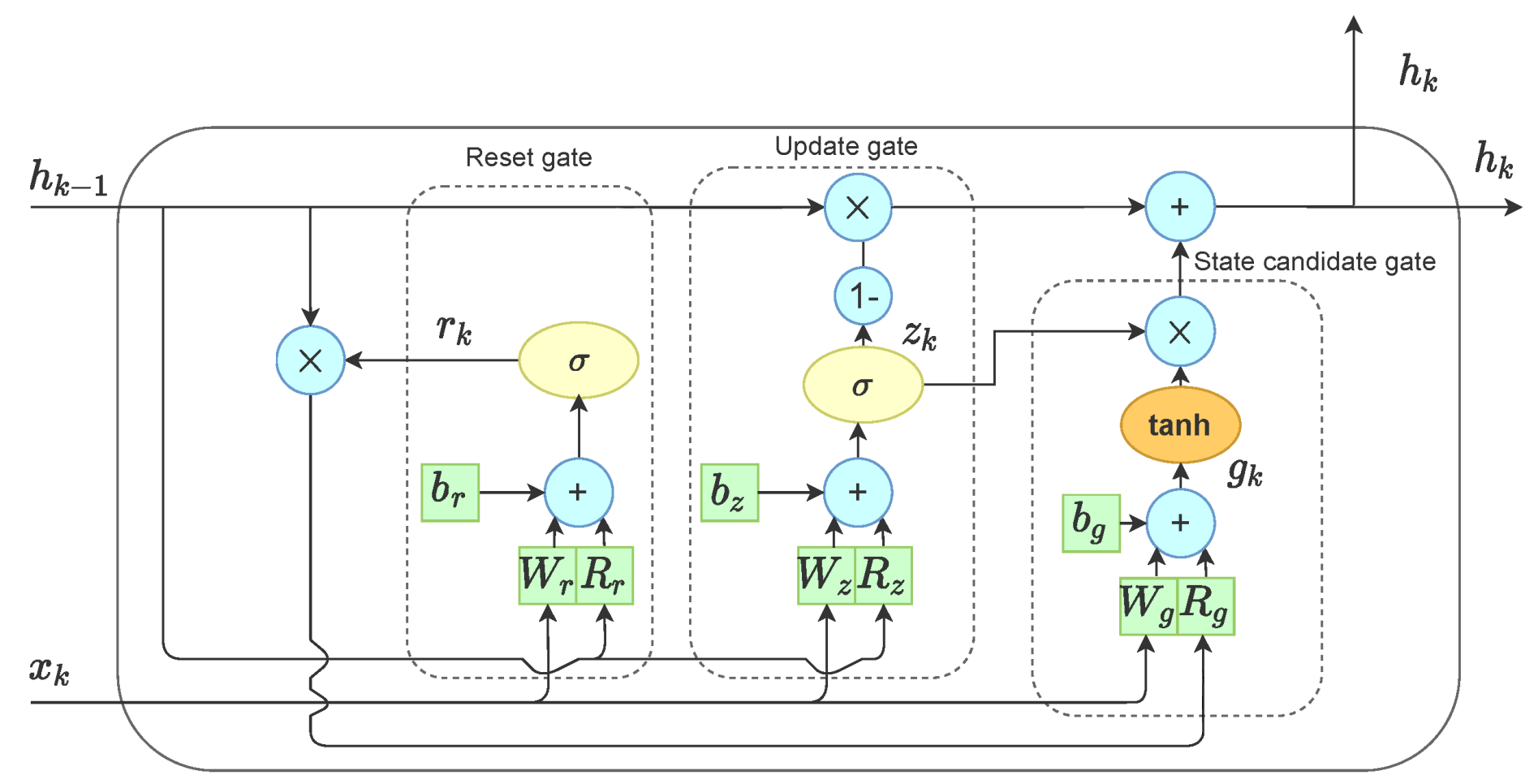
- $z_t = \sigma(W_z x_t + R_z h_{t-1} + b_z)$ (update gate)
- $r_t = \sigma(W_r x_t + R_r h_{t-1} + b_r)$ (reset gate)
- $g_t = \tanh(W_g x_t + R_g(r_t \odot h_{t-1}) + b_g)$ (candidate hidden state)
- $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot g_t$ (new hidden state)

donde $\sigma(\cdot)$ es la función sigmoide y $\odot$ es el producto elemento a elemento (Hadamard). La tanh puede ser reemplazada por otra función activación pero es el estándar.

Doble bias

Tanto la implementación en Pytorch como en Keras posee doble bias.

Gated Recurrent Unit



¿Entendimos los GRUs?

```
model = Sequential([
  layers.Input(shape=(None,1)),
  layers.Bidirectional(layers.GRU(16)),
  layers.Dense(4,activation="softmax")
])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
bidirectional_2 (Bidirectional)		
dense_2 (Dense)		

¿Entendimos los GRUs?

```
model = Sequential([
  layers.Input(shape=(None,1)),
  layers.Bidirectional(layers.GRU(16)),
  layers.Dense(4,activation="softmax")
])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
bidirectional_2 (Bidirectional)	(None, 32)	
dense_2 (Dense)	(None, 4)	

¿Entendimos los GRUs?

```
model = Sequential([
  layers.Input(shape=(None,1)),
  layers.Bidirectional(layers.GRU(16)),
  layers.Dense(4,activation="softmax")
])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
bidirectional_2 (Bidirectional)	(None, 32)	1,824
dense_2 (Dense)	(None, 4)	132